Universidade de Aveiro
2009
Departamento de Electrónica, Telecomunicações e Informática

Luís Miguel Marques da Silva

Desenvolvimento de DSRC segundo a pré-norma IEEE 802.11p

Universidade de Aveiro
2009
Departamento de Electrónica, Telecomunicações e Informática

Luís Miguel Marques da Silva

Desenvolvimento de DSRC segundo a pré-norma IEEE 802.11p

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Dr. João Nuno Pimentel Silva Matos, Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

Presidente	Professor Doutor Nuno Miguel Gonçalves Borges de Carvalho Professor Associado da Universidade de Aveiro
Vogais	Professor Doutor José Carlos dos Santos Alves Professor Associado da Faculdade de Engenharia da Universidade do Porto
	Professor Doutor João Nuno Pimentel Silva Matos Professor Associado da Universidade de Aveiro

agradecimentos

Aos meus pais e irmã, deixo a minha enorme gratidão pela educação que me deram, por toda a atenção, amor e carinho com que marcaram a minha vida e por me deixarem prosseguir os meus sonhos.

Ao amor da minha vida, Liliana, agradeço a tua presença na minha vida e todo o amor e força que me dás.

Gostaria de agradecer ao meu orientador, Prof. Dr. João Nuno Pimentel da Silva Matos, e, Eng. Ricardo Matos Abreu, a oportunidade que me foi facultada ao participar neste projecto.

Agradeço ao Eng. Pedro Mar pelo seu apoio, disponibilidade e pela simpatia prestada ao longo deste trabalho.

A todos os meus amigos que estão comigo desde sempre, agradeço-vos a vossa amizade.

palavras-chave

DSRC, OFDM, VHDL, QUARTUS II, FPGA, Extensão cíclica.

resumo

Esta dissertação aborda o tema da modulação OFDM em sinais digitais, mais concretamente, a adição de uma extensão cíclica entre os símbolos OFDM.
Para este trabalho, é necessária uma síntese de conhecimentos sobre o standard IEEE 802.11a e, nomeadamente, sobre a modulação OFDM bem como o DSRC, VHDL e FPGAs.
A dissertação apresentada pretende dar a conhecer ao leitor noções do uso de novas tecnologias para assegurar a prevenção e melhoramento da circulação rodoviária, concretamente, uso de técnicas de modulação de sinais digitais a fim de obter um melhor rendimento e fiabilidade destes.
Esta dissertação descreve o processo de desenvolvimento de um sistema de adição de um prefixo cíclico entre os símbolos OFDM, em linguagem VHDL, de modo a garantir a melhor prestação na transmissão e recepção de sinais digitais.

keywords

DSRC, OFDM, VHDL, QUARTUS II, FPGA, Cyclic Extension

abstract

This dissertation addresses the theme of OFDM modulation in digital signals, more substantially, the addition of a cyclic extension between OFDM symbols.
For this work, is necessary a synthesis of acquirements about IEEE 802.11a standard and, namely, OFDM modulation as well as DSRC, VHDL and FPGAs.
The thesis described intends to inform the reader about ideas of new technologies use for ensure and improvement the road circulation, substantially, the use of modulation techniques of digital signals with the goal of obtain better efficiency and reliability of them.
This dissertation describes the process of developing a system of a cyclic prefix addition between OFDM symbols, in VHDL language, with the finality of ensure the better fulfillment in transmission and reception of digital signals.

Índice

Índice de figuras

Índice de tabelas

Lista de Siglas

WLAN – *Wireless Local Area Network*

IEEE – *Institute of Electrical and Electronics Engineers*

PHY – *Physical*

MAC – *Media Access Control*

OFDM – *Orthogonal Frequency Division Multiplexing*

Mbps – *Mega bits per second*

PC – *Personal Computer*

PDA – *Personal Digital Assistant*

LAN – *Local Area Network*

GHz – *GigaHertz*

MPDU – *Mac Protocol Data Unit*

PLCP – *Physical Layer Convergence Procedure*

PMD – *Physical Medium Dependent*

QAM – *Quadrature Amplitude Modulation*

EUA – Estados Unidos da América

U-NII - *Unlicensed National Information Infrastructure*

FDM – *Frequency Division Multiplexing*

ICI – *Inter Carrier Interference*

DFT – *Discrete Fourier Transform*

DSL – *Digital Subscriber Line*

SNR – *Signal to Noise Ratio*

FSK – *Frequency Shift Keying*

MCM – *Multi Carrier Modulation*

A/D – Analógico / Digital

IFFT – *Inverse Fast Fourier Transform*

IQ – *In-phase Quadrature*

AWGN – *Additive White Gaussian Noise*

FFT – *Fast Fourier Transform*

BER – *Bit Error Rate*

BPSK – *Binary Phase Shift Keying*

QPSK – *Quadrature Phase Shift Keying*

Km – Quilómetros

KHz – *KiloHertz*

DSP – *Digital Signal Processing*

FT – *Fourier Transform*

ISI – *Inter Symbol Interference*

RF – Rádio Frequência

PAPR – *Peak to Average Power Ratio*

ADSL – *Asymmetric Digital Subscriber Line*

DAB – *Digital Audio Broadcasting*

DVB-T – *Digital Video Broadcasting Terrestrial*

DSRC – *Dedicated Short Range Communications*

RSU – *Road Side Unit*

OBU – *On Board Unit*

CSMA – *Carrier Sense Multiple Access*

EIRP – *Equivalent Isotropically Radiated Power*

AEI – *Automatic Equipment Identification*

ATIS – *Automatic Terminal Information Service*

ASTM – *American Society for Testing and Materials*

VHDL – *(Very High Speed Integrated Circuits) Hardware Description Language*

ASIC – *Application Specific Integrated Circuit*

FPGA – *Field programmable Gate Array*

VHSIC – *Very High Speed Integrated Circuits*

DoD – *Department of Defense*

CLB – *Configurable Logic Block*

IOB – *Input Output Block*

RAM – *Random Access Memory*

SRAM – *Static Random Access Memory*

EPROM – *Erasable Programmable Read Only Memory*

EEPROM – *Electrically Erasable Programmable Read Only Memory*

CPLD – *Complex Programmable Logic Device*

Km/H – Quilómetros Por Hora

AHDL – *Altera Hardware Description Language*

HDL – *Hardware Description Language*

SOPC – Systems On a Programmable Chip

Capítulo I – Introdução

O tráfego rodoviário tem-se tornado, nos últimos anos, um dos principais problemas da população. Os congestionamentos, os acidentes de trânsito, a falta de prevenção de acidentes e alertas de perigo, a perda de tempo com cobranças de portagens e outros impostos, são alguns dos problemas que afectam não só a qualidade de vida como também causam prejuízos, principalmente a nível económico.

Como seria de esperar, ao longo da última década tem sido uma das principais preocupações de grande parte dos países, tentar minimizar estes problemas através de elevados investimentos na área dos transportes, aplicando novas tecnologias na prevenção rodoviária e mecanismos de facilitar a condução nas estradas.

1.1 – Enquadramento do projecto

Com a evolução das tecnologias associadas às telecomunicações, as operações na área dos transportes têm sido cada vez mais viáveis em termos de custo e eficácia, tais como, alertas em tempo real de situações adversas na via pública e ainda a redução de acidentes de trânsito, tempo perdido em congestionamentos, consumo de energia e danos ambientais, contribuindo para a sustentabilidade do sector. Estas tecnologias podem ser utilizadas para:

- Informações de tráfego, informações de acidentes.
- Mapeamento da condição do pavimento, condições meteorológicas, actividades e perigos na via pública.
- Serviços de aviso de emergências (policia, bombeiros).
- Localização automática de veículos.
- Pagamentos de portagens e impostos relacionados com o veículo.

Para um bom desempenho destas tecnologias é necessária a existência de normas que possibilitem a criação e evolução de novas metodologias e equipamentos que viabilizem a realização de projectos. O trabalho aqui descrito está inserido num projecto denominado DSRC 5.9 GHz VII WAVE, cujo objectivo é o desenvolvimento de um protótipo capaz de mostrar as potencialidades das novas tecnologias aliadas às telecomunicações para prevenção rodoviária.

Este projecto possui várias fases de trabalho, consistindo no desenvolvimento das funcionalidades específicas do DSRC e pré-norma IEEE 802.11p. Uma das partes do trabalho é a transmissão e recepção de sinais e,

respeitando as normas, é de grande importância que o seu desempenho seja fiável pois estão em causa possíveis acidentes na via pública. Segundo as normais mais evoluídas das telecomunicações o uso da técnica de modulação OFDM possui muitas vantagens em comparação com outras modulações, o que torna este sistema de transmissão/recepção, mais eficaz e menos propenso a erros.

1.2 – Motivação

Tem sido realizada investigação no âmbito das tecnologias aplicadas à prevenção rodoviária no sentido de desenvolver sistemas de prevenção e informação de tráfego. Com o constante crescimento do tráfego rodoviário tem-se verificado um aumento significativo de perigos e sinistralidade, o que leva a um aumento de congestionamentos, acidentes e todas as situações resultantes de tal conjuntura [1].

Para tal têm sido desenvolvidos sistemas destinados a minimizar a sinistralidade na via pública, com todas as vantagens daí esperadas, devendo respeitar normas internacionais e utilizar técnicas de envio e recepção dos sinais que permitam tirar o máximo rendimento e performance possível.

Este é um desafio que aceitei participar ao longo deste trabalho.

1.3 – Objectivos

O principal objectivo desta dissertação é contribuir para a implementação de um sistema de transmissão/recepção que garanta fiabilidade e segurança na transmissão de dados por radiofrequência. A norma IEEE 802.11p [2] indica a utilização da modulação OFDM.

O trabalho realizado incide sobre o desenvolvimento de uma parte pertencente ao sistema OFDM que efectua a adição de uma extensão cíclica entre os blocos OFDM. Esta extensão cíclica é o alvo deste projecto sendo a sua realização um passo importante no sistema OFDM, pois garante estabilidade e eficiência da informação entre o transmissor e receptor.

Este sistema é desenvolvido em linguagem VHDL, com base na ferramenta de software Quartus II v.9.0 Web edition, da Altera. Esta ferramenta de desenvolvimento também permite a simulação do software desenvolvido de forma a garantir e provar a correcta adição da extensão cíclica.

1.4 – Estrutura da dissertação

Após uma breve introdução no início do presente capítulo, o enquadramento e a fundamentação do trabalho a ser realizado, bem como a definição dos objectivos que esta dissertação se propõe, surgem os capítulos seguintes:

- **Capítulo II – Introdução teórica** que começa por explorar conceitos relacionados com a norma IEEE 802.11 e as suas características. Será dado especial ênfase à técnica de modulação OFDM. O capítulo continua com a descrição das características e aplicações do DSRC. Por fim será focada a linguagem VHDL que é a base para a implementação do sistema, a história e enquadramento das FPGAs e ainda algumas referências à empresa Altera e o seu software Quartus II.

- **Capítulo III – Conceptualização do sistema** que apresenta um método para o desenvolvimento do sistema e suas partes constituintes bem como uma descrição generalista de cada um dos seus blocos.

- **Capítulo IV – Implementação e resultados do sistema** que descreve em termos gerais o modo como o protótipo foi implementado e mostra os resultados das simulações funcionais de cada bloco e sistema geral. O capítulo divide-se em oito partes: a *vista geral do sistema*, o bloco *selector*, *fifo_control*, *fifo_control2*, *fifo*, *fifo2*, *lpm_shiftreg* e *or*.

- **Capítulo V – Conclusões e Trabalho Futuro** que é dedicado à descrição das conclusões após a elaboração desta dissertação. No final são também apresentadas sugestões para trabalho futuro.

Capítulo II – Introdução teórica

2.1 – Tecnologia Wireless

As tecnologias wireless, cada vez mais nos dias de hoje, têm-se tornado populares na área dos negócios e nas nossas vidas pessoais. Através de PCs e/ou PDAs podemos aceder a serviços de correio electrónico, internet, listas telefónicas calendários, etc. As empresas, particulares e agências têm utilizado e massificado o uso de tecnologias wireless (WLANs) nos seus ambientes de trabalho.

As tecnologias wireless são simples de serem usadas e tornam possível um ou mais dispositivos comunicarem sem recorrerem ao uso de conexões físicas (cabos) o que liberta os espaços de trabalho de cabos indesejados e facilita a portabilidade dos dispositivos que acedem à rede tornado o trabalho mais eficiente, reduzindo custos e aumentando a produtividade.

Os principais riscos inerentes à tecnologia wireless são a perda de conectividade e a possível falta de segurança, ou seja, a invasão por parte de terceiros podendo dados importantes/confidenciais serem perdidos ou corrompidos [3].

2.2 – Normas Wireless

As WLANs são baseadas na norma IEEE 802.11 que foi desenvolvida a partir de 1997.

Esta norma foi criada para suportar transmissões wireless de médio alcance com velocidades de transmissão de dados elevada, entre dispositivos móveis ou entre estes e estações de trabalho. Originalmente a norma 802.11 foi desenhada para transmissões wireless entre 1 a 2 Mbps tendo sido posteriormente actualizada para várias versões com taxas de transmissão mais elevadas em diferentes bandas de frequência [3].

2.3 – Norma IEEE 802.11a

A norma IEEE 802.11a é uma versão da original IEEE 802.11, e foi ratificada em 1999 usando o mesmo núcleo da norma IEEE 802.11.

Esta norma especifica que a camada física (PHY) utilize a modulação OFDM, que separa a informação do sinal pelas portadoras, operando na faixa dos

5 GHz e utiliza 52 portadoras (OFDM) com uma taxa máxima de transmissão teórica de 54 Mbps o que na prática e realisticamente atinge uma taxa máxima de transmissão na ordem dos 20-30 Mbps. Esta taxa pode ser reduzida se assim se pretender para 48, 36, 24, 18, 12, 9 e 6 Mbps. Nesta norma as taxas de 6, 12 e 24 Mbps estão por defeito sempre presentes sendo as principais do conjunto. Quatro portadoras do conjunto de 52 são denominadas portadoras piloto e são usadas pelo sistema como referência para desfasamentos ou desvios de frequência durante a transmissão do sinal. As restantes 48 portadoras providenciam caminhos separados para o envio dos dados de modo paralelo. O espaçamento em frequência existente entre as portadoras é de 0.3125 MHz (para uma largura de banda de 20MHz com 64 possíveis portadoras) [3].

Tabela 1: Parâmetros da camada PHY [4]

Taxa de transferência de dados	6, 9, 12, 18, 24, 36, 48 e 54 Mbit/s
Modulação	BPSK OFDM QPSK OFDM 16 – QAM OFDM 64 – QAM OFDM
Código corrector de erros	K=7 (64 estados) código convolucional
Taxa de codificação	1/2, 2/3, 3/4
Número de portadoras	52
Duração do símbolo OFDM	4 µs
Intervalo de guarda	0.8 µs
Largura de banda ocupada	16.6 MHz

Também na norma 802.11a, o primeiro objectivo da OFDM PHY é transmitir as unidades de dados (MPDUs) provenientes da camada MAC. A camada PHY é dividida em dois elementos principais: a subcamada PLCP e a subcamada PMD. A camada MAC comunica com a subcamada PLCP através de um serviço na PHY por primitivas específicas. Quando a camada MAC dá ordem de transmitir, o PLCP prepara os MPDUs para essa transmissão e também envia frames para a camada MAC. A subcamada PLCP minimiza a dependência da

camada MAC na subcamada PMD através do mapeamento dos MPDUs numa frame formatada adequadamente para transmissão.

A subcamada PMD promove a transmissão e recepção entre as camadas PHY de duas estações, através do meio wireless. Para realizar este intento, a subcamada PMD interage directamente com o meio e promove a modulação e desmodulação dos frames a transmitir. A subcamada PLCP e PMD comunicam usando um serviço de primitivas para assegurar as funções de transmissão e recepção [3].

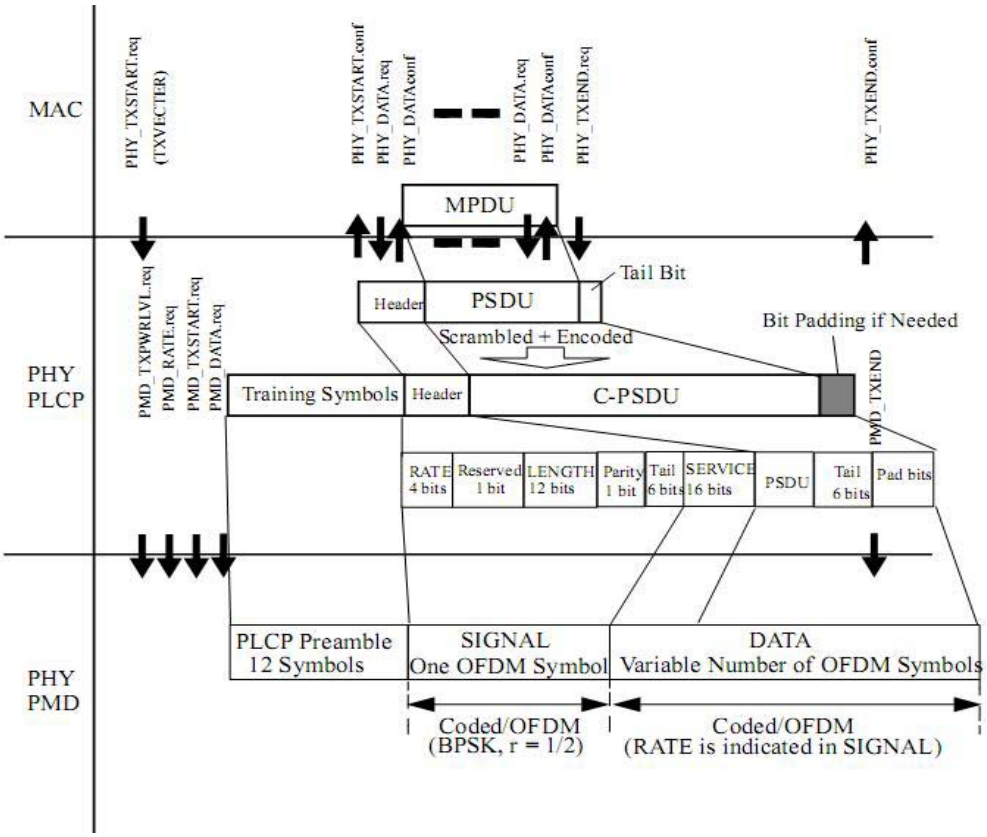

Figura 1: Procedimento de transmissão da subcamada PLCP [3]

Com a modulação OFDM um sinal série binário é dividido, sendo representado em símbolos de um, dois, quatro ou seis bits dependendo da taxa de transmissão escolhida, e é convertido em números complexos representando uma constelação de pontos.

Por exemplo, se a taxa de transmissão escolhida for 24 Mbps, então o PLCP mapeia os bits de dados para uma constelação 16QAM [3].

Tabela 2: Parâmetros de modulação da norma IEEE 802.11a [4]

Taxa de dados (Mbit/s)	Modulação	Taxa de codificação (R)	Bits codificados por portadora (N_{BPSC})	Bits codificados por símbolo OFDM (N_{CBPS})	Bits de dados por símbolo OFDM (N_{DBPS})
6	BPSK	1/2	1	48	24
9	BPSK	3/4	1	48	36
12	QPSK	1/2	2	96	48
18	QPSK	3/4	2	96	72
24	16 - QAM	1/2	4	192	96
36	16 - QAM	3/4	4	192	144
48	64 - QAM	2/3	6	288	192
54	64 - QAM	3/4	6	288	216

Tabela 3: Parâmetros temporais OFDM PHY [4]

Parâmetro	Valor
N_{SD}: número de portadoras de dados	48
N_{SP}: número de portadoras piloto	4
N_{ST}: Número total de portadoras	52 (N_{SD}+N_{SP})
Δ_F: Espaçamento de frequência entre portadoras	0.3125 MHz (20 MHz/64)
T_{FFT}: Período IFFT/FFT	3.2 µs (1/ Δ_F)
$T_{PREAMBLE}$: Duração preâmbulo PLCP	16 µs (T_{SHORT} + T_{LONG})
T_{SIGNAL}: Duração do símbolo BPSK OFDM	4.0 µs (T_{GI} + T_{FFT})
T_{GI}: Duração intervalo de guarda	0.8 µs (T_{FFT}/4)
T_{GI2}: Duração intervalo de guarda do símbolo de treino	1.6 µs (T_{FFT}/2)
T_{SYM}: Intervalo de símbolo	4 µs (T_{GI} + T_{FFT})
T_{SHORT}: Duração sequência de treino curta	8 µs (10 x T_{FFT}/4)
T_{LONG}: Duração sequência de treino longa	8 µs (T_{GI2} + 2 x T_{FFT})

Nos EUA, estão alocados 300 MHz de largura de banda na banda dos 5 GHz para as WLANs sob as regras da U-NII. Esta largura de banda está fragmentada em dois blocos que são descontínuos na banda dos 5 GHz [5].

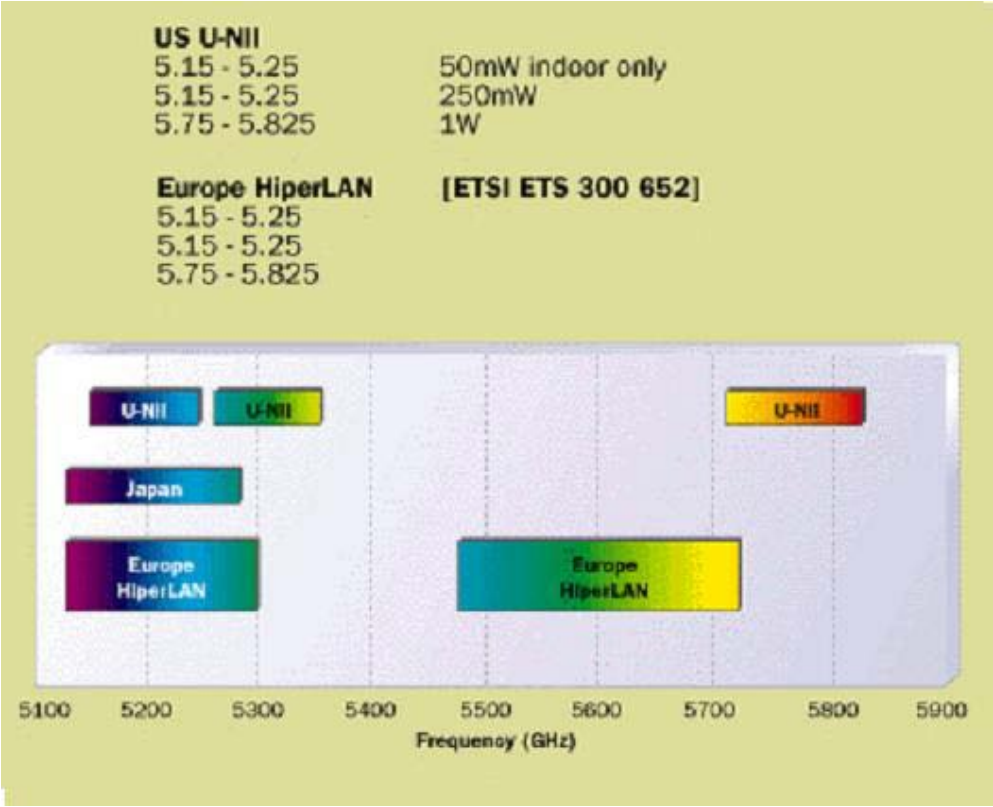

Figura 2: Alocação do espectro de frequência na banda dos 5 GHz [5]

Trabalhando na banda dos 5 GHz é oferecida maior largura de banda do que na banda de 2.4 GHz pelo que, também é menos susceptível a interferências já que a banda de 2.4 GHz é dividida com mais aplicações wireless tais como o Bluetooth. Por outro lado existem pequenos detalhes a considerar tais como o facto da banda de 5 GHz não ter uma frequência contínua e ainda a potência de transmissão dos dispositivos tem que ser mais elevada do que na banda de 2.4 GHz [5].

2.4 – OFDM

2.4.1 – História OFDM

O uso do conceito OFDM nas comunicações desenvolveu-se do uso da multiplexagem na frequência (FDM). Acerca de um século atrás, a utilização do telégrafo fez com que vários sinais de baixa taxa de transmissão fossem conduzidos num canal usando diferentes portadoras a diferentes frequências ocupando uma determinada largura de banda. Estas portadoras estão espaçadas o suficiente para não se sobreporem.

Este conceito faz com que a interferência entre canais (ICI) seja minimizada. Este sistema comparado com os sistemas de telecomunicações dos dias de hoje é muito ineficiente, mas através das pesquisas e desenvolvimentos feitos durante as décadas de 50 e 60, os sistemas evoluíram ao nível da eficiência de transmissão de dados paralelos e técnicas de FDM [6].

Tabela 4: Breve história do OFDM [7]

Ano	Evento
1966	Chang mostra que a modulação multi-portadora resolve o problema de multipath sem reduzir a taxa de transmissão. Este conceito foi considerado a primeira publicação oficial de modulação multi-portadora.
1971	Weinstein e Ebert mostram que a modulação multi-portadora pode ser executada usando a DFT.
1985	Cimini trabalhando nos laboratórios Bell identificou muitos dos pontos essenciais na transmissão OFDM e implementou um desenho do seu conceito.
1993	DSL adopta o OFDM, também denominado discrete multi-tone.
1999	O comité IEEE 802.11 wireless LANs aprovou o standard 802.11a para OFDM na banda 5GHz UNI.
2002	O comité IEEE 802.16 aprovou o standard baseado em OFDM para acesso às redes wireless para áreas metropolitanas como revisão ao standard 802.16a.
2003	O comité IEEE 802.11 aprovou o standard 802.11g para operar na banda 2.4GHz.
2003	O standard multi-banda OFDM para ultra Wideband é desenvolvido, mostrando a utilidade do OFDM em sistemas de baixo SNR.

2.4.2 – Sistema OFDM

Com este novo conceito, a intenção é que as portadoras se sobreponham. Isto só será possível de ser executado, se as portadoras possuírem um sinal cuja taxa de transmissão esteja espaçada de um múltiplo de 1/T na frequência, em que T é o período do sinal. Os sinais são ortogonais se forem mutuamente independentes uns dos outros [6].

A ortogonalidade é uma propriedade que permite que a informação de vários sinais seja transmitida num canal e detectada sem interferências. A perda desta propriedade resulta na sobreposição da informação dos sinais provocando perda e degradação de informação nas comunicações [8].

Usando este critério, a largura de banda disponível é usada de um modo muito mais eficiente enquanto é prevenido o aparecimento de ruído. Esta eficiência pode preservar 50% da largura de banda [6].

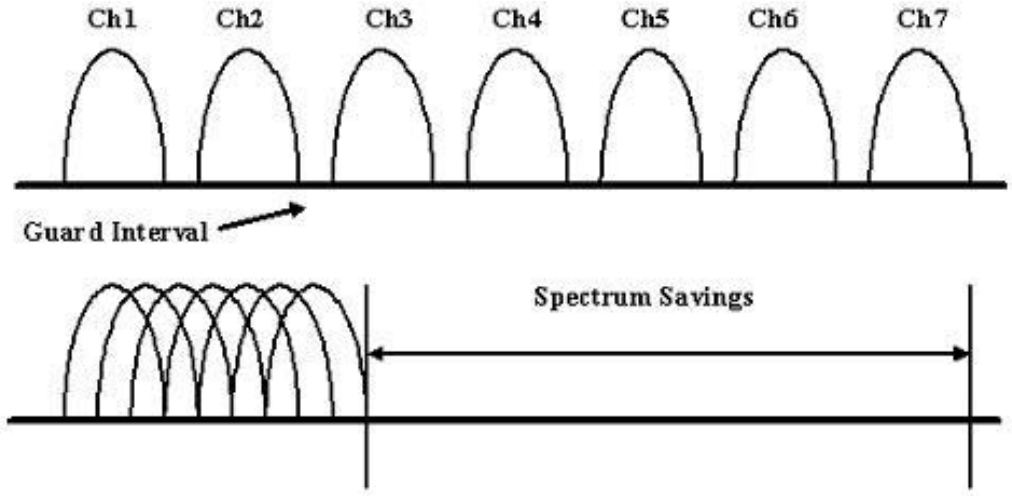

Figura 3: Alocação do espectro de frequência no OFDM [6]

Este tipo de comunicação híbrida, provém da combinação dos conceitos de Multi-Carrier Modulation (MCM) e Frequency Shift Keying (FSK). No entanto este tipo de comunicação cria problemas ao receptor quando este processa os dados.

Uma maneira de prevenir este problema é garantir que os dados que são extraídos no receptor possuam todas as portadoras ortogonais. Para máxima eficiência um determinado número de portadoras vai dividir o espectro disponível. Isto faz com que seja permitido o transporte de dados paralelamente em vez do transporte série.

Usando um determinado número de portadoras permite um maior número de taxas, no lugar de uma única taxa. O benefício é uma largura de banda usada menor e menos propensão a ruído ou interferência entre canais. A resposta do canal torna-se mais linear quanto menor for a largura de banda [6].

Um sistema de comunicação é constituído por cinco componentes principais: *fonte*, *transmissor*, *sentido*, *canal* e *receptor* [6].

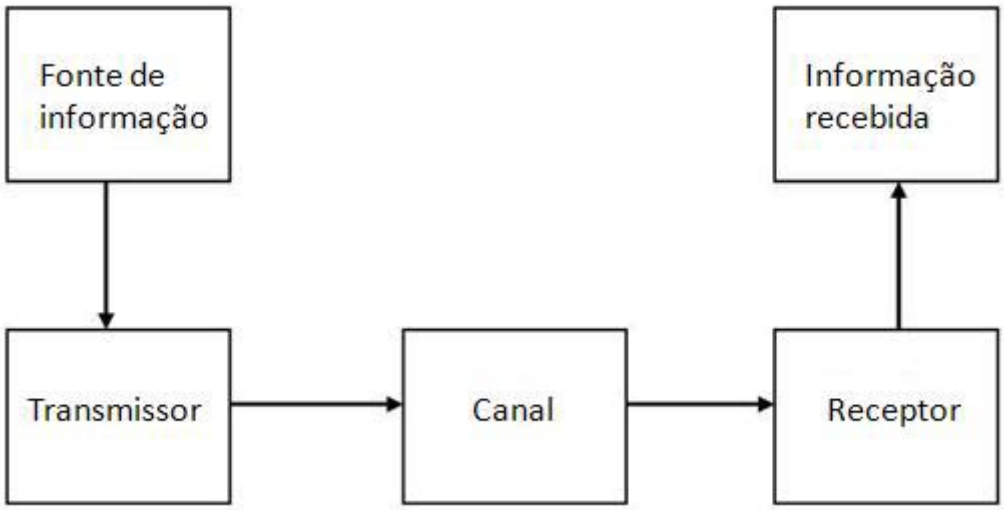

Figura 4: Sistema de comunicação digital típico [7]

A fonte aplica a sua informação ao transmissor. Dependendo do tipo de sistema, digital ou analógico, a informação pode ser processada de modo a garantir que esteja no formato correcto para o transmissor. O transmissor envia a informação através do canal, enquanto o receptor recolhe a informação adicionada de ruído introduzido pelo canal. O receptor passa a informação para o destino onde vai ser convertido no tipo de sinal apropriado para o seu uso. Para um sistema de comunicação wireless o ar ou espaço representa o seu canal.

A transição através do meio wireless não é tão estável como nas transmissões com fios, e isto representa problemas que precisam de ser resolvidos no receptor, para garantir que a informação é descodificada de um modo correcto. A modulação OFDM é usada principalmente em sistemas de comunicação digitais, embora se o sinal de entrada for analógico, a informação é convertida num sinal digital através dum conversor A/D. Este último sinal é colocado num transceiver OFDM para produzir os sinais OFDM.

No transmissor o sinal é codificado geralmente por um codificador convolucional. O codificador produz um determinado número de bits de saída por cada bit de entrada, dando redundância ao sinal. De seguida o sinal codificado seguirá para o interleaver que reduzirá o impacto de erros por sobreposição de bits movendo bits contíguos para posições não contíguas. Posteriormente o sinal é modulado para símbolos no bloco de modulação sendo posteriormente os dados separados em vários streams em número igual ao número de portadoras, a isto denomina-se série para paralelo.

Aqui são adicionados bits piloto aos símbolos. Este conjunto é posteriormente enviado ao bloco IFFT. Seguidamente é efectuada a operação inversa da série – paralelo, ou seja, é efectuada a operação paralelo – série. De seguida o bloco extensão cíclica adiciona uma extensão aos símbolos, em que é copiado um determinado número de portadoras do final de cada símbolo e são movidas para o início desse símbolo. Por fim o sinal é convertido para analógico por um modulador IQ sendo transmitido para o canal que introduz ruído AWGN e dispersão. O sinal agora está corrompido e o objectivo do receptor é extrair a informação desejada do sinal recebido. Como o canal introduz ruído e offset na fase e frequência, o receptor necessitará de ajustar estas imperfeições usando o bloco de sincronização. A extensão cíclica também é removida, seguindo-se o bloco série – paralelo. O número de streams é igual ao número de portadoras do transmissor. Os streams de dados são desmodulados para símbolos OFDM usando o bloco FFT. Os símbolos resultantes são convertidos de novo de paralelo para série e são removidos também os bits piloto. Estes bits são comparados com outros para determinar diferenças de fase, frequência e amplitude. Os sinais temporais resultantes são usados para determinar os sinais de dados a serem desmodulados. O sinal é convertido num stream binário no bloco desmodulador, sendo seguido do bloco de-interleaver. Aqui os dados resultantes estão na ordem correcta igual aos dados originais. Estes dados são enviados para o bloco descodificador onde é extraída a informação original [6].

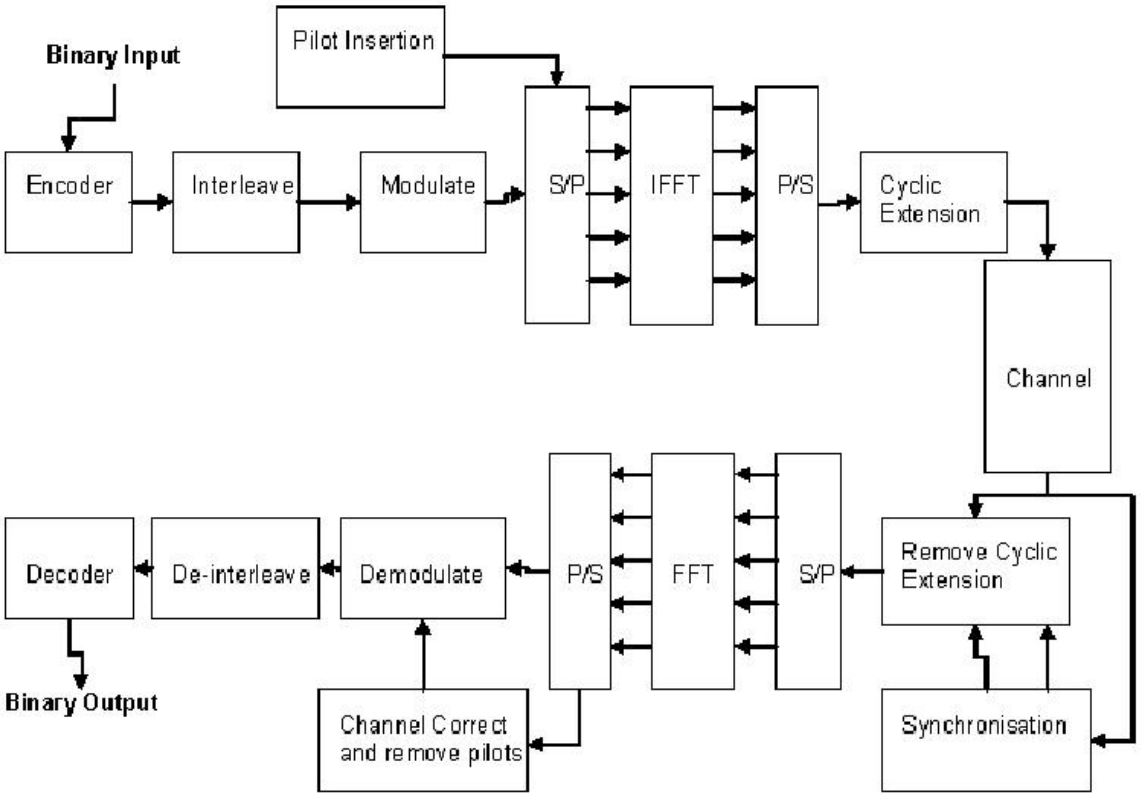

Figura 5: Diagrama de blocos de um transceiver [6]

2.4.2.1 – Codificador

A utilização de um codificador que controle os erros torna todo o sistema muito mais eficiente e viável. É capaz de reduzir a taxa de erros (BER) para uma dada quantidade de ruído no sinal (SNR). O custo para o sistema é que, é introduzida redundância no sinal, resultando numa redução do rendimento efectivo. Dependendo da taxa de codificação escolhida, a representação original dos bits é aumentada de acordo com a taxa dada. Isto produz um aumento do número de bits a transmitir comparativamente ao número de bits originais do sinal. A taxa de codificação é definida pelo número de bits de entrada, Ko, a dividir pelo número dos bits de saída, No. A taxa pode ser expressa pela expressão:

$$R = \frac{\mathrm{Ko}}{\mathrm{No}}$$

Geralmente existem diferentes taxas de codificação para diferentes cenários. As taxas de codificação usadas em sistemas de comunicação OFDM são 1/2, 2/3 ou 3/4. Quando é utilizada a taxa 1/2, dois bits na saída representam

um bit original de entrada. Isto representa um aumento de duas vezes o tamanho do sinal original [6].

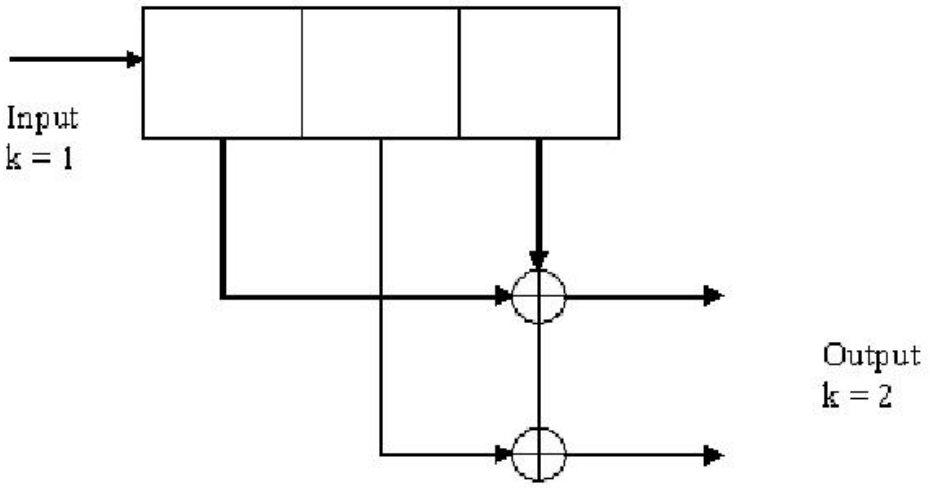

Figura 6: Diagrama de codificador convolucional OFDM [6]

Para a utilização de uma taxa 2/3, a taxa de 1/2 contém alguns bits omitidos. Quando dois bits são codificados, perfazendo quatro bits de saída, um dos bits adicionados é excluído deixando três bits representar os dois bits originais de entrada. Para a taxa de 3/4 o mesmo princípio é usado. Seis bits de saída representam três bits de entrada mas dois bits de saída são omissos. Os restantes quatro bits representam os três bits de entrada. A omissão daqueles bits é uma operação chamada de “puncturing” [6].

2.4.2.2 – Interleaver

A função de um bloco interleaver nos sistemas de comunicação OFDM é evitar os efeitos de desvanecimento do sinal e respostas não lineares que afectam o espectro de frequência. As portadoras OFDM ocupam diferentes frequências que fazem com que as amplitudes das portadoras também sejam diferentes.

Para reduzir o impacto da diferença de amplitudes no stream codificado, os bits são separados em zonas que se localizem contiguamente aquando do seu mapeamento para as respectivas portadoras. Assim, o stream terá um pequeno espaçamento entre bits no lugar de um grande bloco, se ocorrer alguma interferência ou ruído. Nos sistemas OFDM geralmente é usado um bloco interleaver M*N. Os bits são escritos no bloco nas colunas e lidos por linhas [6].

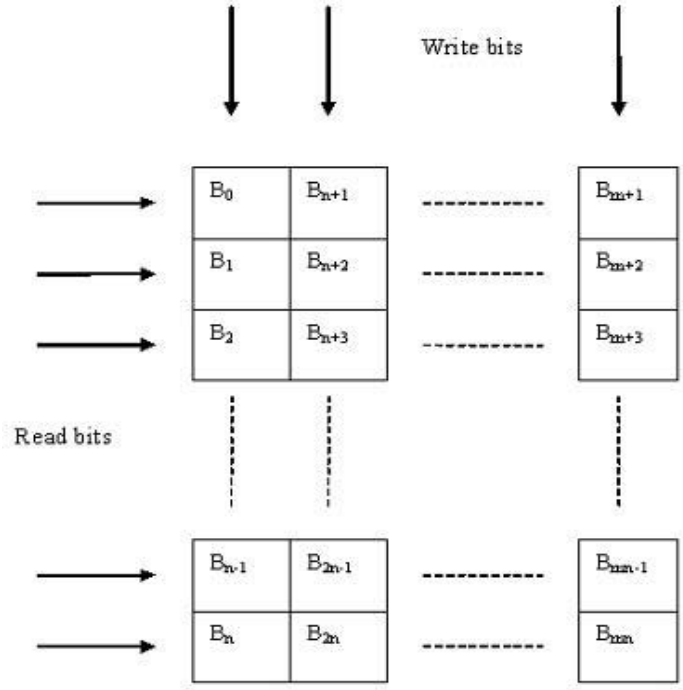

Figura 7: Bloco interleaver [6]

2.4.2.3 – Modulador

O sinal codificado é modulado em símbolos para aumentar a eficiência do sistema de comunicação. Esta modulação dos sinais altera a amplitude, fase e frequência do sinal. Com o OFDM, apenas a fase e amplitude é variável. A situação e aplicação definem o tipo de modulação escolhida. Através da conversão dos bits em símbolos, os números complexos representam um ou mais bits, dependendo da modulação escolhida.

Nos sistemas de comunicação OFDM os esquemas de modulação são o BPSK, QPSK, 16 – QAM e 64 – QAM. Cada modulação mapeia um determinado número de bits num símbolo. Isto pode ser visto através do mapa de constelação de cada tipo de modulação. Para o BPSK, um bit representa um símbolo enquanto no QPSK, dois bits correspondem a um símbolo. No 16 – QAM existem quatro bits que representam um símbolo e no 64 – QAM existem 6 bits por símbolo.

A taxa de erro (BER), aumenta para o mesmo nível de taxa de ruído no sinal (SNR) tal como o mapeamento de bits por símbolo aumenta. A taxa de ruído necessita ser minimamente elevada para no receptor a remoção dos bits seja efectuada correctamente. As pequenas diferenças de fase que cada tipo de modulação possui quando o número de pontos da constelação aumenta fazem

com que a taxa de ruído também aumente. Se o número de pontos da constelação aumenta a potência média da constelação também aumenta segundo a equação:

$$Pave = \frac{1}{\mathrm{M}} \sum_{\mathrm{k}=1}^{\mathrm{M}} |\mathrm{Ck}|^2$$

Onde: **M** é o número de pontos na constelação.

Ck é a potência de todos os pontos M da constelação.

Da fórmula acima é fácil verificar da relação directa entre número de pontos e potência do sinal. Tal como a potência do sinal varia com o número de pontos, a probabilidade de erro num bit também varia. A equação para a probabilidade de aparecer um bit errado é:

$$Pb = Q\left(\sqrt{\frac{Eb}{No}}\right)$$

Onde: **Q** é a função densidade probabilidade.

Eb é a energia do bit.

No é a potência do ruído.

Para o BPSK, existem apenas dois resultados para “0” e para “1”. Estes resultados encontram-se separados por 180º. A constelação do BPSK é [6]:

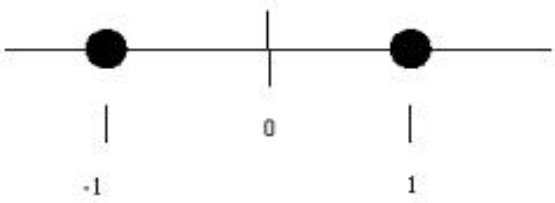

Figura 8: Constelação BPSK [6]

Quando os dados atravessam através dum canal ruidoso, os pontos não se encontram exactamente nos pontos da constelação acima. A diferença entre os pontos exactos e os pontos que são recebidos depois do canal é chamada de vector de erro. Com o BPSK, não existem factores imaginários como parte de números complexos. Isto faz com que o resultado esteja no eixo real, um num lado ou outro da marca zero. O receptor decide qual o ponto que é suposto ser, ou “0” ou ” 1” dependendo de qual lado estiver em relação ao zero.

Para o QPSK, existem quatro pontos na constelação. Cada ponto é constituído por uma parte real e uma parte imaginária que perfaz o número complexo. Isto quer dizer que, existem pontos em ambos os lados do zero no eixo real bem como existem pontos dos dois lados do zero no eixo imaginário. A área onde o ponto se encontra, depois de afectado pelo ruído, é apenas um quarto da constelação se este for analisado correctamente. Se o ponto tiver uma variação de fase superior a 90^{o}, ele vai aparecer num quadrante diferente e o receptor vai interpretar como se fosse um ponto diferente e vai ocorrer um erro. Para garantir que os erros são minimizados, a taxa de ruído necessita de ser tão grande como o esquema BPSK. O receptor vai decidir de entre quatro pontos qual será em comparação à escolha de um ponto de dois possíveis na modulação BPSK. A constelação do QPSK será então [6]:

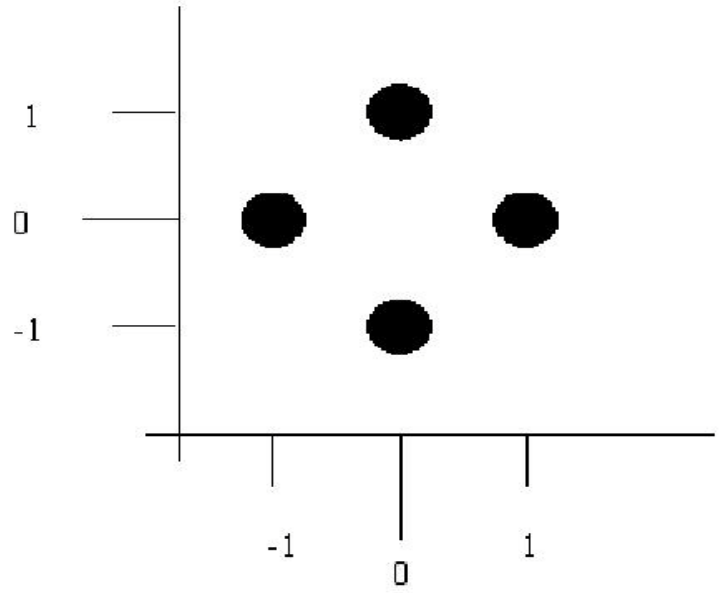

Figura 9: Constelação QPSK [6]

A modulação 16 – QAM tem dezasseis pontos, que são os necessários para o receptor garantir que os dados estão correctos. Isto permite apenas uma variação de fase de 45º, antes de ocorrer um erro. Tal como a fase do sinal varia para representar os diferentes pontos, a modulação QAM, também altera a amplitude do sinal. Se a fase ou amplitude variarem, pode resultar num erro. A constelação 16 – QAM é [6]:

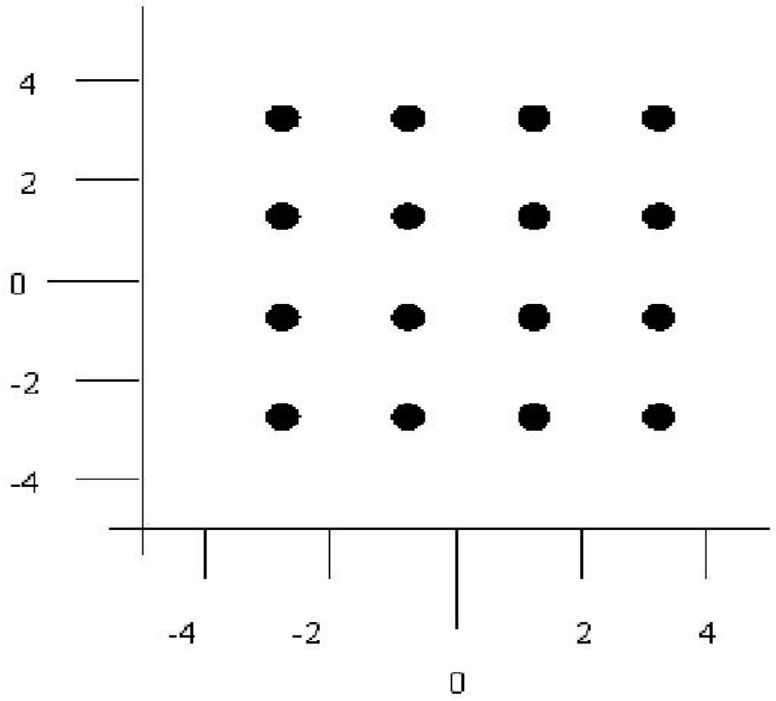

Figura 10: Constelação 16 – QAM [6]

Finalmente, na modulação 64 – QAM existem sessenta e quatro pontos na constelação. A diferença de fase que a modulação permite antes de ocorrer um erro é cerca de 22.5º. Este, valor é metade da modulação 16 – QAM, então os pontos precisam ser mais precisos para garantir que o receptor decifra correctamente o sinal. Existem quatro níveis de amplitude nesta modulação. O sinal necessita de uma taxa de ruído maior para garantir que os dados são extraídos correctamente do sinal. A constelação da modulação 64 – QAM é a seguinte:

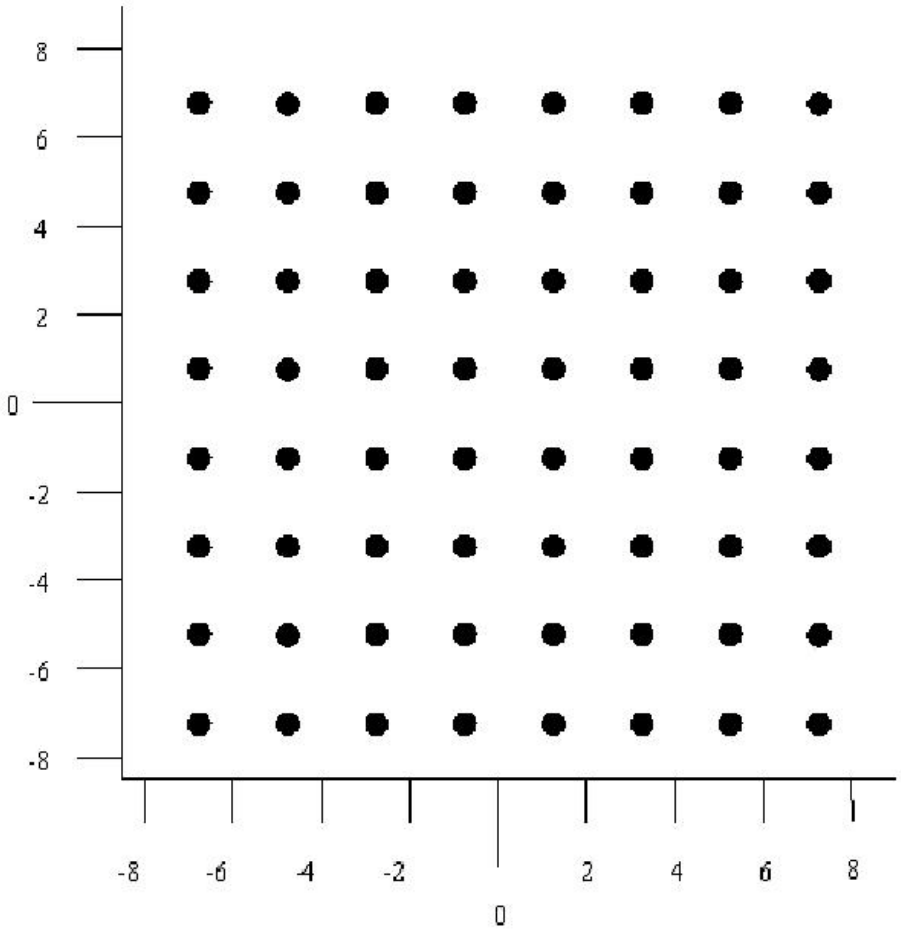

Figura 11: Constelação 64 – QAM [6]

2.4.2.4 – Inserção de bits piloto

O sinal transmitido, passando pelo canal wireless, é afectado por um desfasamento em fase e frequência bem como por ruído. Para ajudar o receptor a extrair a informação do sinal recebido, são inseridos bits piloto no sinal OFDM. Estes bits estão posicionados por todo o símbolo OFDM para ter um efeito máximo na ajuda ao receptor na detecção de variações de fase e/ou frequência do sinal transmitido.

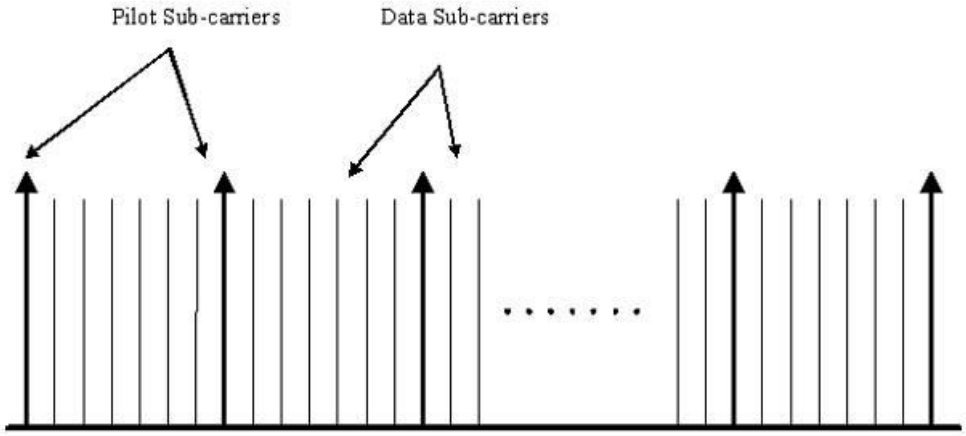

Figura 12: Bits piloto e bits de dados [6]

A sincronização no receptor do sinal transmitido permite ao receptor extrair a informação desejada. O espaçamento dos bits piloto depende muito do tipo de canal wireless, a sua largura de banda e o seu tempo de coerência. Quanto mais ruído tiver o canal, mais ajuda e tempo o receptor necessitará para sincronizar o sinal recebido.

Os bits piloto não transportam qualquer informação útil. O conjunto de bits piloto torna-se um compromisso entre a performance do canal e a taxa de ruído (SNR). O atraso devido ao processamento dos bits piloto abranda o sistema e o seu rendimento. Através do conhecimento do atraso máximo, τ_{max} o menor espaçamento entre os bits piloto pode ser calculado. Este espaçamento tem que ser menor que o atraso máximo para garantir a integridade dos dados.

$$Espaçamento = \frac{1}{\tau max}$$

Onde: τ_{max} é o atraso máximo.

Tomando por referência uma estação móvel típica cuja área de cobertura é aproximadamente 5Km, o atraso máximo é o round trip time a dividir pela velocidade da luz.

$$Atraso = \frac{(5000 * 2)}{3 * 10^8} = 33.33\ \mu s$$

Com um atraso de 33µs, o espaçamento mínimo é ≈ 30KHz e tendo o conhecimento do espaçamento de frequência das portadoras, a posição e número dos bits piloto podem ser encontrados. Os bits piloto têm dados com números complexos de valores 1 + j0 e -1 + j0. Sendo assim, os bits piloto apenas têm apenas parte real.

Estes valores são guardados no receptor com intenção de comparação. Através da comparação dos bits piloto no receptor com os recebidos do pacote OFDM, é feita uma estimativa das diferenças de fase, frequência tempo e amplitude. Esta estimativa pode ser usada para compensar as alterações do sinal [6].

2.4.2.5 – **Transformada de Fourier**

Com a análise dos sistemas de comunicação, é muito importante perceber a relação que os sinais têm no domínio da frequência e no domínio do tempo. Muitos sistemas estudados no domínio da frequência são fornecidos à entrada um sinal aleatório e variável no tempo. Uma das ferramentas usadas para providenciar essa relação é a transformada de Fourier (FT). Dado um sinal de entrada periódico e complexo com período To:

$$x(t) = Ae^{j2\pi fot}$$

Onde: **A** é a amplitude do sinal.

fo é a frequência do sinal em Hz.

O sinal ortogonal é representado pelos coeficientes da série de Fourier. Os coeficientes da série de Fourier do sinal x(t) são:

$$\mathrm{x_n} = \frac{1}{To} \int_{\alpha}^{\alpha+To} x(t)\, e^{-j2\pi t \frac{n}{To} dt}$$

Onde: **To** é o período do sinal.

α é uma constante indicativa do limite inferior.

x(t) é o sinal de entrada.

O termo:

$$\frac{n}{To} = nfo$$

Onde: **To** é o período do sinal.

fo é a frequência do sinal.

Os coeficientes representam os harmónicos da frequência fundamental. Os coeficientes são geralmente valores complexos não tendo em consideração que os sinais podem ser complexos ou reais [6].

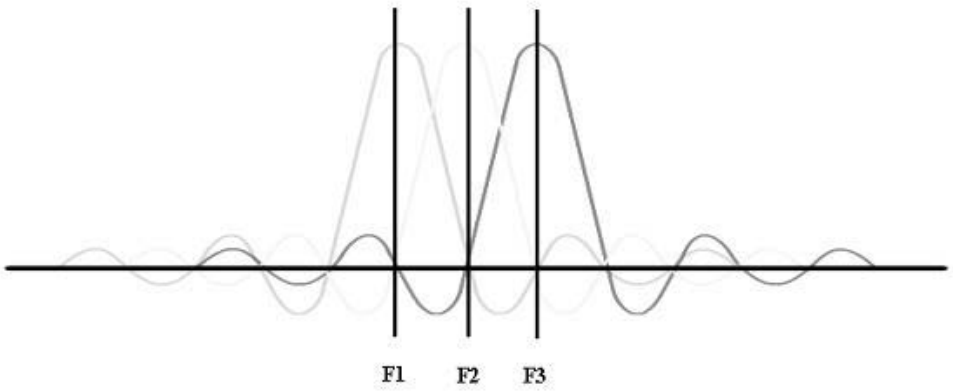

Figura 13: Portadoras ortogonais [6]

2.4.2.5.1 – IFFT

Uma vez colocados os bits piloto nos símbolos, os dados são enviados para a IFFT. Aqui os símbolos complexos de dados são mapeados para o domínio do tempo. Os símbolos OFDM e outra informação temporal necessária são transportados nas portadoras ortogonais dentro da banda de frequência requerida. Nem todas as portadoras são usadas para a informação de dados e bits piloto.

Existem algumas que são utilizadas como portadoras de guarda e outras como preâmbulo no início e fim dos símbolos OFDM. Para o uso ser o mais eficiente possível da IFFT, o número de portadoras é tido como uma potência de dois, 2^n. Anteriormente à entrada dos processadores digitais do sinal (DSPs), os

quais permitem à IFFT ser usada num único chip, com um banco de misturadores de sinais, osciladores e filtros que constituem esta função. Este conjunto de equipamento significa que o tamanho é um factor limitativo neste tipo de comunicações [6].

2.4.2.6 – **Extensão cíclica**

Na modulação OFDM, são usadas altas taxas de transmissão de dados, e para isto acontecer de um modo eficiente, existem alguns itens a serem cumpridos para garantir que a eficiência é alta. Um destes itens é a introdução de um intervalo de guarda entre os símbolos OFDM, que seja maior que o atraso introduzido pelo canal, de modo a minimizar a interferência entre símbolos.

Este intervalo de guarda pode ser introduzido na forma de uma extensão cíclica. Esta extensão cíclica pode ser efectuada segundo duas operações denominadas prefixo cíclico e sufixo cíclico, em que no primeiro caso uma última secção de cada símbolo OFDM é copiada para o início formando um prefixo, e no segundo caso uma primeira secção de cada símbolo é copiada para o fim formando um sufixo. No caso do OFDM a técnica geralmente mais usada é o prefixo cíclico.

Se o atraso for maior do que a extensão cíclica, então, a interferência entre símbolos (ISI) pode ocorrer, fazendo com que os símbolos OFDM se sobreponham e se perca a ortogonalidade das portadoras provocando interferência entre portadoras (ICI), tornando o sistema inoperável [6].

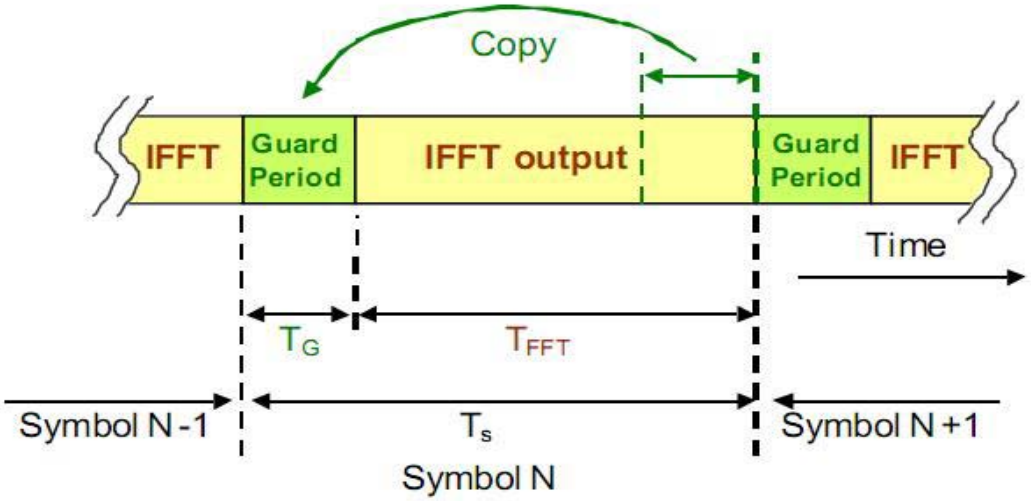

Figura 14: Extensão cíclica – Prefixo cíclico [8]

A eficiência e taxa de ruído do sinal podem ser comprometidas se o tamanho da extensão cíclica for demasiado elevado. Os dados pertencentes à extensão cíclica não vão ser usados como informação relevante, fazendo com que o sistema perca eficiência. A energia transmitida pela extensão cíclica adiciona ruído, levando a uma redução da SNR. O tamanho deve então ser escolhido com cuidado e de acordo com o meio para se conseguir obter a melhor performance possível. Em sistemas com blocos de informação longos (> 128 portadoras), a extensão pode ser tão pequena como 10% de cada bloco [6].

O efeito de propagação multipath, é causado pela reflexão, da transmissão do sinal via rádio, em objectos tais como paredes, edifícios, montanhas, etc. Estes múltiplos sinais chegam ao receptor em tempos diferentes devido às diferentes distâncias. Isto faz com que os limites dos símbolos se estendam provocando perda de energia entre eles.

Num sinal OFDM a amplitude e fase da portadora mantém-se constante no período do símbolo para manter a ortogonalidade entre portadoras. Se elas não forem constantes isto significa que a forma do espectro das portadoras não possui a forma duma sinc, e por conseguinte os nulos não se encontram nas frequências correctas, resultando em ICI. Nos limites dos símbolos a amplitude e fase pode mudar rapidamente para o novo valor do símbolo seguinte. Em ambientes multipath, a ISI causa perda de energia entre símbolos, resultando em mudanças de amplitude e fase no início de cada símbolo.

A duração destes efeitos corresponde ao atraso do canal. O sinal transitório é o resultado de cada componente que chega a diferentes tempos, mudando o vector recebido da cada portadora. A figura 14 mostra este efeito. Adicionando um intervalo de guarda dá tempo para a transição do sinal para decair, fazendo com que a FFT, comece a partir duma porção do símbolo correcta. Os restantes efeitos causados pelo fenómeno de multipath, tais como rotação de fase e picos de amplitude são corrigidos pela equalização do canal [8].

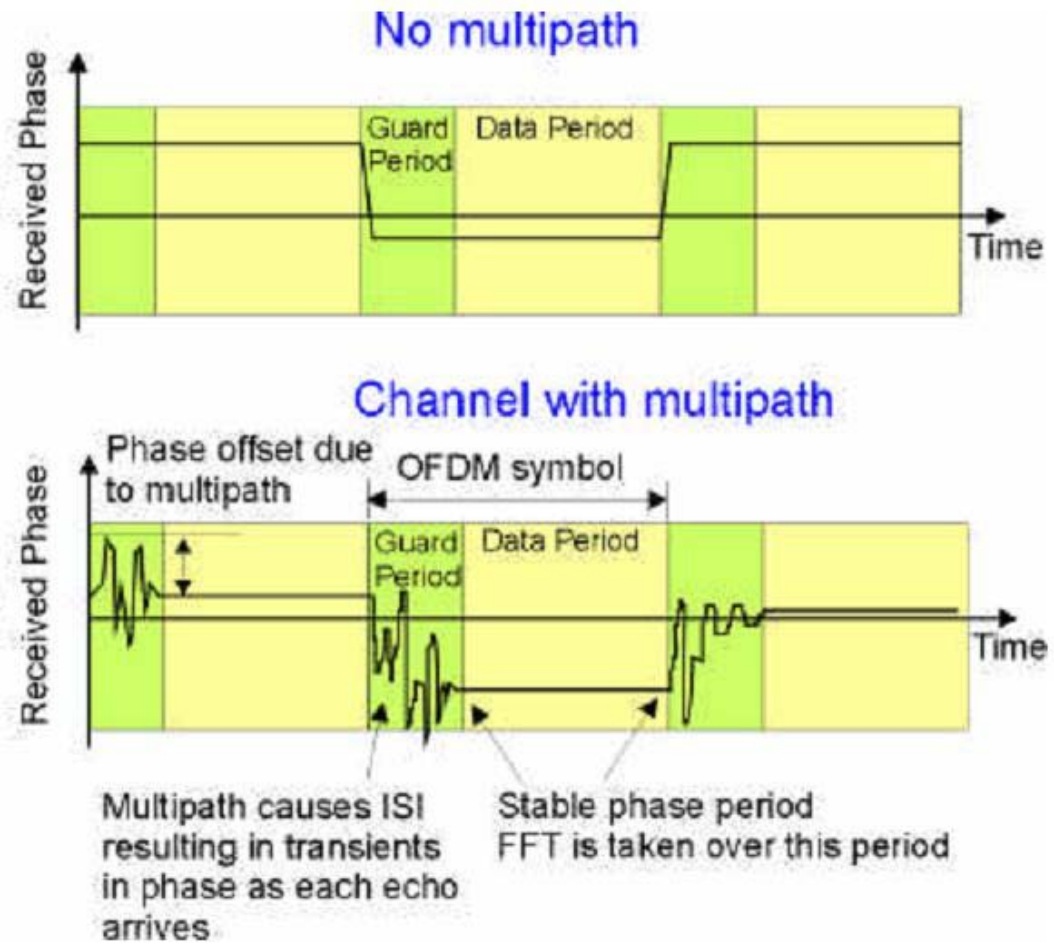

Figura 15: Protecção contra a ISI em ambientes multipath através da inserção do intervalo de guarda [8]

2.4.2.7 – Transmissão

Agora os símbolos OFDM estão num sinal série, modificado para analógico e convertido para a frequência requerida para a transmissão. O símbolo OFDM pode ser escrito como:

$$s(t) = \mathcal{R}\left\{ \sum_{i=-\frac{Ns}{2}}^{\frac{Ns}{2}-1} Xi\,,k\; e^{j2\pi[fc+\frac{i}{Tfft}(t-kT)]} \right\},kT\ \leq t \leq kT+T$$

Onde: **Ns** é o número de portadoras.

Xi, k são os dados complexos e sinal piloto.

Fc é a frequência central do OFDM.

i é o índice das portadoras.

k é o índice dos símbolos transmitidos.

T é o período do símbolo OFDM.

Tfft é o período efectivo do símbolo OFDM.

O sinal é emitido através dum canal wireless e uma sequência contínua OFDM é transmitida podendo ser expressa pela expressão:

$$Srf\ (t) = \sum_{-\infty}^{\infty} Srf,k\ (t-kT)$$

Para a extracção correcta da informação do sinal, o receptor vai ter em conta todos os efeitos de alterações que afectam o sinal [6].

2.4.2.7.1 – Efeito Doppler

A frequência do sinal recebido pode variar em função da velocidade relativa entre o transmissor e o receptor. Nas comunicações móveis, o utilizador está geralmente em movimento de aproximação ou afastamento das estações rádio. Isto afecta a frequência do sinal recebido. É como um motociclista que se

aproxima de nós, ou seja, o som do motor é diferente do que quando ele se afasta.

A frequência do motor é maior quando se aproxima e menor quando se afasta. Esta diferença na frequência é chamada de efeito de Doppler e tem que ser levada em conta aquando da sincronização no receptor do sinal transmitido. A variação de frequência causada pelo efeito de Doppler é [6]:

$$\Delta f = \frac{2v}{\lambda}$$

Onde: **v** é a velocidade em metros por segundo.

λ é o comprimento de onda em metros.

Na modulação OFDM, este conjunto de alterações na frequência podem causar a perda da ortogonalidade das portadoras o que provocaria o aparecimentos de erros na transmissão de dados [6].

2.4.2.8 – **Recepção**

O receptor implementa o processamento inverso efectuado no transmissor com algumas funções extra adicionais. O receptor necessita de estar sincronizado na frequência e fase com o sinal recebido de modo a garantir que os dados são extraídos correctamente. Depois do andar de radiofrequência, o sinal é convertido à sua banda – base e então o receptor procura por uma sequência pré – definida ou preâmbulo. Isto define os limites de cada transmissão OFDM. Uma vez encontrados os limites, o receptor determina o desvio na frequência. Esta operação é efectuada através do estimador de canal usando os bits piloto nos símbolos OFDM.

A extensão cíclica é removida ainda antes da operação FFT. A função FFT é utilizada e as correcções implementadas ao sinal. Assim neste momento o sinal e o receptor encontram-se sincronizados um com o outro. O sinal é desmodulado, passando ao de-interleaver e finalmente ao descodificador. O resultado final é um stream digital de bits igual ao original [6].

2.4.3 – Vantagens e desvantagens do OFDM

2.4.3.1 – Vantagens

O sistema OFDM possui várias vantagens comparativamente a outro tipo de técnica de modulação implementada nos sistemas wireless.

2.4.3.1.1 – Aproveitamento da largura de banda

Um dos aspectos mais importantes nas comunicações é a eficiência no aproveitamento da largura de banda disponível. Isto é especialmente importante nas comunicações wireless, onde todos os dispositivos usados nos dias de hoje e no futuro, estão a partilhar um espectro de frequência muito saturado. No OFDM todos os canais estão próximos o suficiente, o que permite o uso mais eficiente da largura de banda. O facto das portadoras serem ortogonais faz com que 50% de largura de banda possa ser preservada comparativamente a outras modulações [9].

2.4.3.1.2 – Eliminação da ISI

Uma das grandes limitações do envio de dados a altas taxas de transmissão é o efeito de interferência entre símbolos. Com a inserção de intervalos de guarda entre os símbolos, está garantida a integridade dos símbolos ao nível do receptor.

2.4.3.1.3 – Imunidade ao ruído

Os sistemas OFDM apresentam grande robustez ao ruído impulsivo, devido ao aumento da duração dos símbolos no tempo. Caso hajam erros provocados por ruído de rajada, os símbolos podem ser recuperados na recepção por meio de esquemas apropriados de codificação, como o uso de técnicas de interleaving [14].

2.4.3.2 – Desvantagens

2.4.3.2.1 – PAPR

Uma grande desvantagem do OFDM é o problema de surgirem picos de potência. A envolvente complexa do sinal OFDM pode apresentar altas excursões e picos de amplitude. O sinal OFDM em banda – base é formado pela soma de M sinais complexos modulados em diferentes frequências. Em alguns casos, estes sinais poder-se-ão somar ou anular em fase, resultando num valor alto para a PAPR (Peak- Average Power Ratio) do sistema. Neste cenário, o amplificador de potência do transmissor irá introduzir distorções não lineares que por sua vez irão destruir a ortogonalidade entre portadoras [14].

2.4.3.2.2 – Desvanecimento do sinal

No OFDM, verifica-se a incompatibilidade do uso do esquema convencional de transmissão em canais selectivos em frequência com desvanecimento, uma vez que a informação transmitida numa portadora pode ser perdida, na presença de um grande desvanecimento. Para combater estes efeitos são usados métodos de codificação combinados com técnicas de interleaving [14].

2.4.3.2.3 – Sincronização

Uma das limitações do OFDM em muitas aplicações é a sensibilidade a erros de frequência causados por diferenças de frequência entre os osciladores locais no transmissor e no receptor. Os offsets na frequência das portadoras levam a defeitos como atenuação e rotação entre as portadoras provocando ICI.

Em ambientes de comunicações móveis o movimento relativo entre transmissor e receptor causa efeito de Doppler e isto faz com que as portadoras não estejam perfeitamente sincronizadas. Esta falta de sincronismo provoca a perda da ortogonalidade que sua vez provocará ICI [9].

2.4.4 – Aplicações do OFDM

O OFDM foi escolhido para vários sistemas de comunicação dos dias de hoje e do futuro em todo o mundo. O OFDM é escolhido principalmente para sistemas onde o canal de transmissão possui características que tornam difícil manter a performance adequada para a comunicação. Conjuntamente com aplicações wireless de alta taxa de transmissão, os sistemas com fios, tais como, *asynchronous digital subscriber line* (ADSL) e modems por cabo utilizam a modulação OFDM.

Nos últimos anos, o OFDM foi adoptado na Europa em várias aplicações de comunicação wireless, tais como, a radiodifusão digital de audio (DAB) e transmissão digital de vídeo terrestre (DVB-T). Ainda mais recentemente o OFDM foi adoptado no WIMAX, sendo este baseado na norma IEEE 802.16, que é uma norma destinada para redes sem fios metropolitanas [9].

2.5 – DSRC

2.5.1 – Conceito DSRC

O DSRC (Dedicated Short Range Communications) é um serviço de comunicação, na banda de 5.9 GHz destinado entre o pequeno e médio alcance, que suporta operações de segurança pública rodoviária bem como operações privadas entre veículos e entre a via pública e os veículos. O DSRC tem em vista ser um complemento de comunicações móveis, providenciando altas taxas de transmissão em circunstâncias onde a minimização da latência e isolamento relativo a pequenas comunicações são importantes [10].

2.5.2 – Características do DSRC

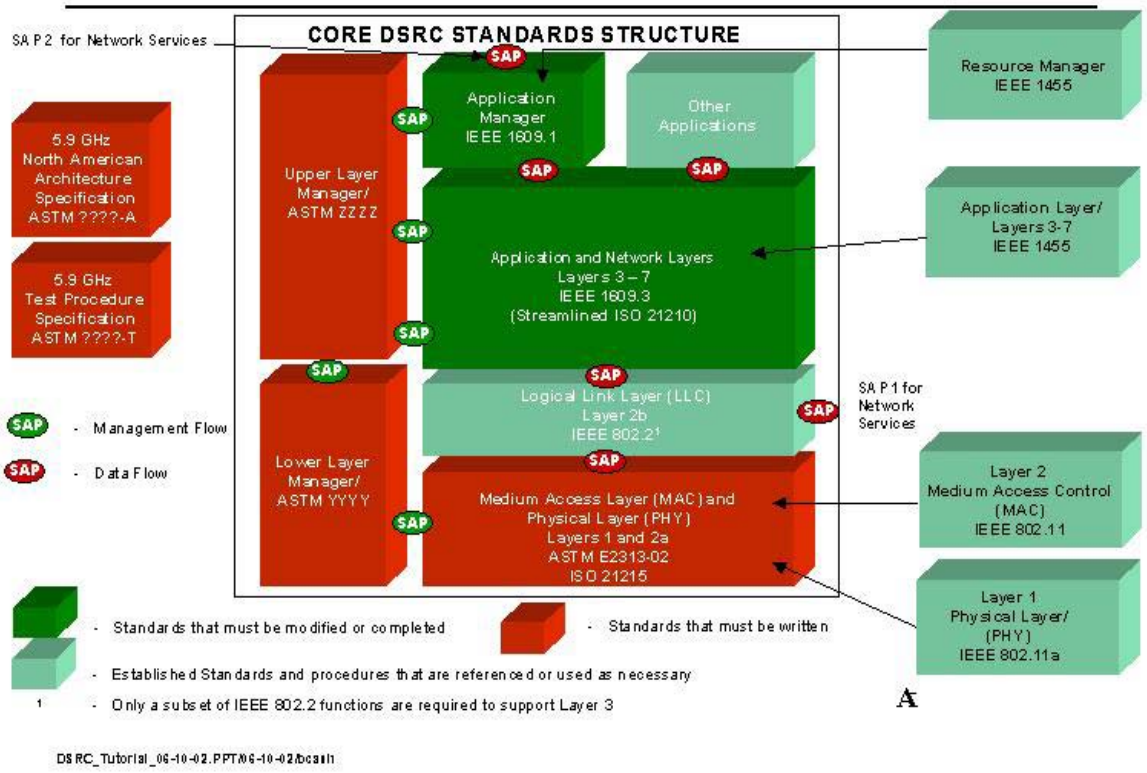

Figura 16: Diagrama estrutural dos standards constituintes do DSRC [10]

Tabela 5: Características tecnológicas do DSRC [10]

Abordagem	Activa
Largura de banda	75 MHz (5.850 - 5.925 GHz)
Modulação	BPSK, QPSK, 16 - QAM e 64 - QAM
Canais	Canais de 7 - 10 MHz (combinações opcionais de canais de 10 e 20 MHz)
Taxa de transmissão	6, 9, 12, 18, 24, e 27 Mbps com canais de 10 MHz 6, 9, 12, 18, 24, 36, 48, e 54 Mbps com canais de 20 MHz
Potência máxima de transmissão	28.8 dBm (entrada da antena)
RSU EIRP	Nominal 0 - 33 dBm (1 mW - 2 W) / Max. 44.8 dBm (30 W)
OBU EIRP	Nominal 0 - 20 dBm (1 - 100 mW) / Max. 44.8 dBm (30 W)
Sensibilidade OBU E RSU	- 82 dBm (QPSK) / - 65 dBm (64QAM)
Interferência do canal	4 - 6 dB (para QPSK @ 10-4 BER coded) / 16 - 17 dB (para 64QAM @ 10-4 BER coded)
	Técnica de partilha de banda – coordenação de frequência. Selecção ou alternância de canais para zonas adjacentes. Uso do CSMA para prevenir interferência entre utilizadores no canal.

Tabela 6: Comparação de capacidades do DSRC em diferentes zonas espectrais [10]

Parâmetros	Banda 902 – 928 MHz	Banda 5850 – 5925 MHz
Espectro usado	12 MHz (909.75 to 921.75 MHz)	75 MHz**
Taxa de transmissão	0.5 Mbps	6 Mbps - 27 Mbps**
Cobertura	Uma comunicação zonal em cada espaço de tempo	Zonas de comunicação sobrepostas**
Estado de alocação	Sem protecção	Estado principal** (alta protecção)
Potenciais interferências	Telefones de 900 MHz Leitores AEI Muitos dispositivos que dispersam o espectro Radares	Radares Militares localizados espaçadamente Satélites de Uplinks localizados muito espaçadamente
Alcance máximo	100 m	1000 m**
Separação mínima	500 m* (excepto onde é plano)	15 m**
Capacidade de canais	1 ou 2 canais	7** canais
Potência (Downlink)	Nominalmente menor que 40 dBm (10 W)	Nominalmente menor que 33 dBm (2 W)
Potência (Uplink)	Nominalmente menor que 6 dBm (< 4mW)	Nominalmente menor que 33 dBm (2 W)

Legenda: * Limitação substancial **Vantagem substancial

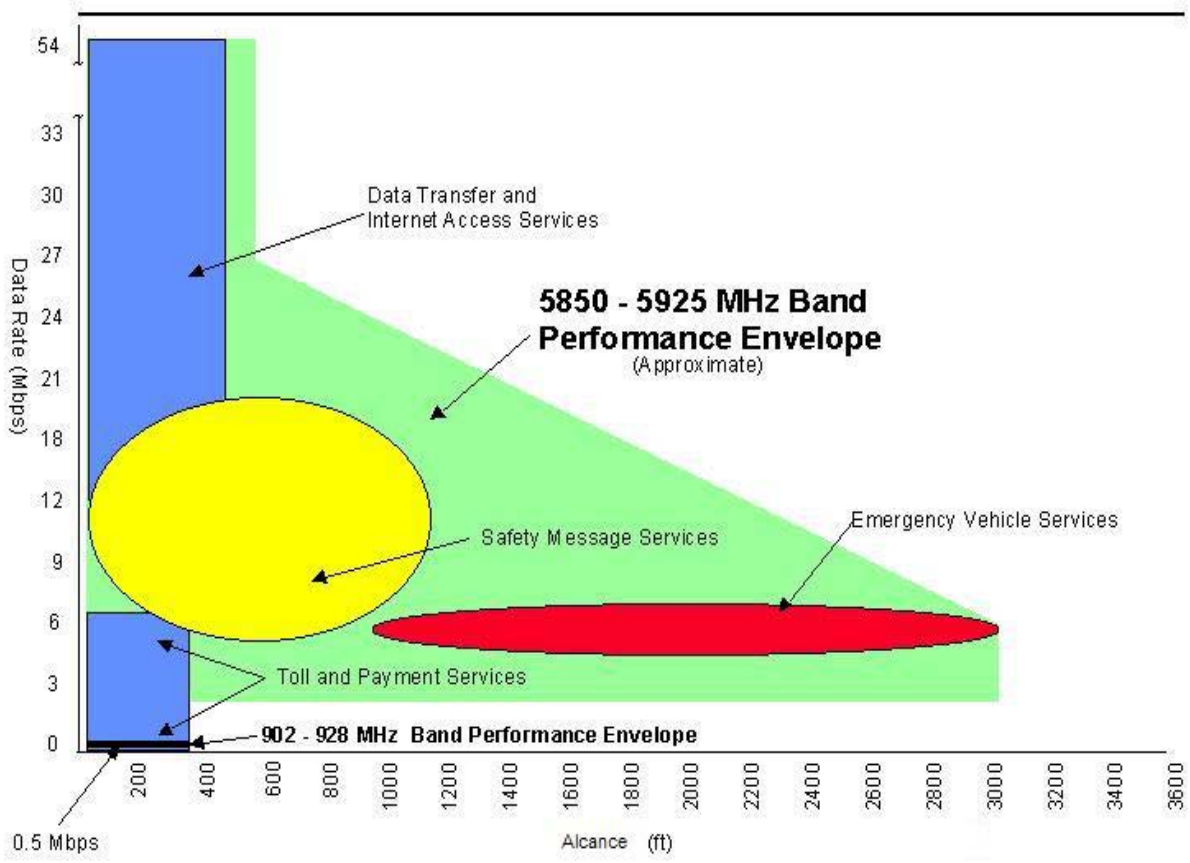

Figura 17: Áreas de performance do DSRC [10]

2.5.2.1 – Características básicas do DSRC

- Canais de 10 MHz.
- Limite RSU EIRP de 44.8 dBm (Segurança pública), 33 dBm (Privado).
- Limite OBU EIRP de 44.8 dBm (Segurança pública), 33 dBm (Privado).
- Emissão fora de canal – 25 dBm (Todos os dispositivos).
- Canal de controlo dedicado para anúncios e avisos.
- As transmissões no canal de controlo obedecem ao standard ASTM/IEEE XXXX.
- Um canal dedicado é reservado para comunicações veículo para veículo.
- Operações de aplicações de intersecção são conduzidas num canal dedicado.
- 2 pequenas zonas nos canais de serviço são designadas para transferências de dados.
- 2 zonas médias nos canais de serviço são designadas para transferências de dados.
- Os canais na banda UNII podem ser usados como canais de serviço sem licença.
- OBUs seguem as instruções dos RSU nos canais de serviço.
- OBUs implementam um limite de tempo nas transacções no canal de serviço [10].

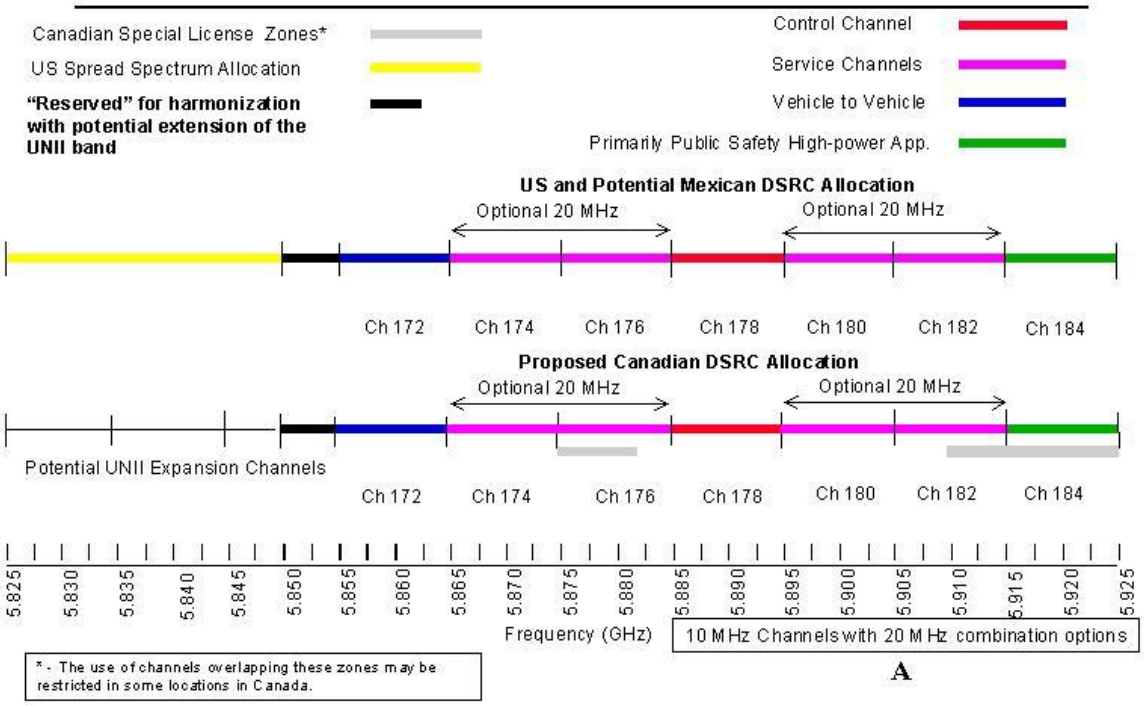

Figura 18: Plano de ocupação de largura de banda do DSRC [10]

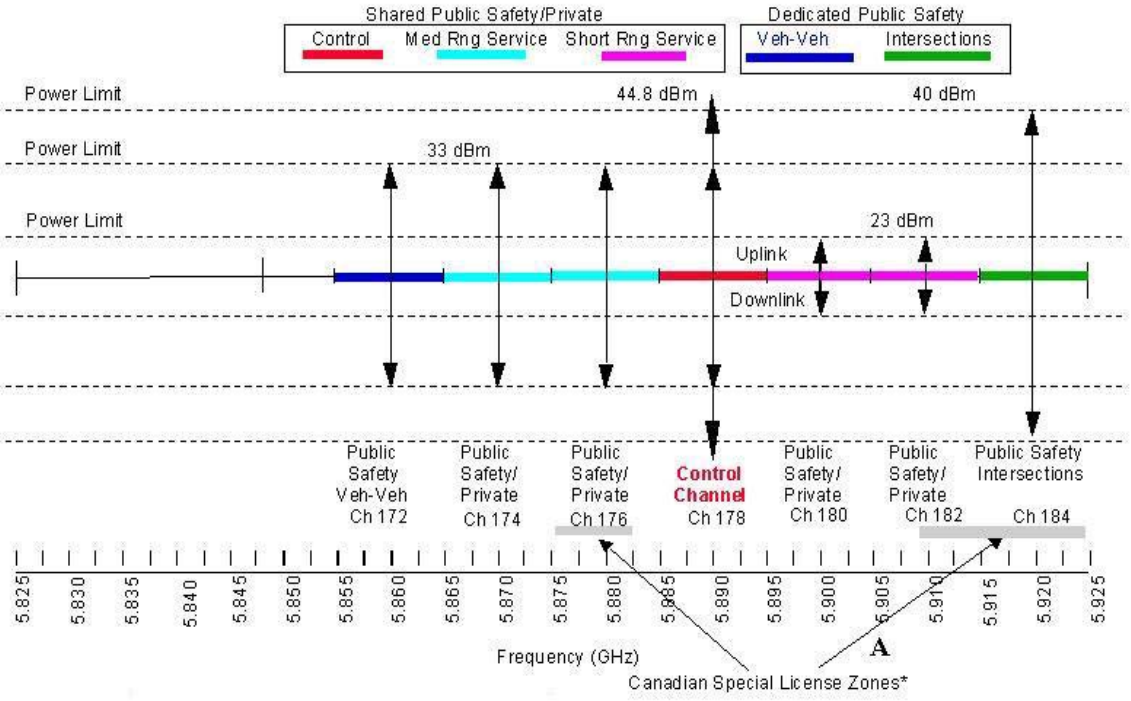

Figura 19: Plano de ocupação de largura de banda com canais de 10 MHz e limites de potência [10]

2.5.3 – Aplicações do DSRC

2.5.3.1 – Segurança pública:

- Assistente de aproximação de veículo de emergência (aviso).
- Prioridade do sinal de veículos de emergência.
- Aviso da condição da via pública.
- Aviso de ponte baixa.
- Aviso de zona de trabalho/obras.
- Aviso de colisão iminente.
- Assistente de velocidade numa curva.
- Infra-estrutura baseada num assistente de luz de travagem.
- Aviso/impedimento de uma colisão de intersecção.
- Impedimento de colisão numa auto – estrada ou caminho-de-ferro.
- Aviso de colisão (veículo para veículo).
- Aviso de velocidade óptima.
- Cruise control adaptável.
- Informação de trânsito.
- Transferência de dados de trânsito entre veículos.
- Sinal de veículos de trânsito prioritários.
- Transmissão de vídeo de veículos de emergência.
- Projecção da via principal.
- Passeios desimpedidos.
- Inspecção do veículo.
- Diário do condutor [10].

2.5.3.2 – **Dados privados:**

- Controlo de acesso.
- Pagamento em condução.
- Pagamento em parques de estacionamento.
- Transferência de dados / informação do nível de combustível.
  - Dados dos ATIS.
  - Dados de diagnóstico.
  - Gravação de serviços de manutenção/reparação.
  - Actualização de programas computacionais do veículo.
  - Actualização de dados de música.
  - Carregamento de vídeos.
- Planeamento de rotas.
- Processamento de aluguer de veículos.
- Controlo rápido e único de operações de veículos comerciais.
- Monitorização do gasto de combustível [10].

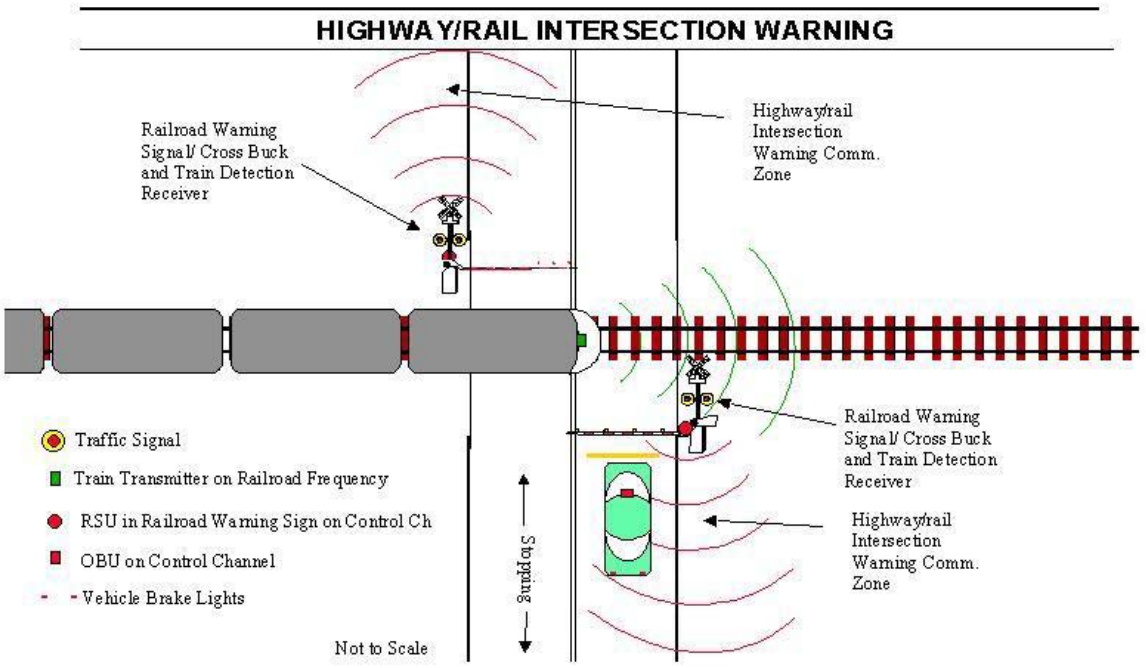

Figura 20: Aviso de intersecção com linha de comboio [10]

2.5.3.3 – Aplicações veículo para veículo

- Veículos com capacidade de comunicações veículo para veículo (v2v) transmitem a posição do veículo, velocidade, direcção, e aceleração a 12 Mbps. Uma transmissão é enviada a cada 300 ms. Esta transmissão é entendida por todos os veículos dentro de um tempo de 10 s, sendo assim a potência transmitida (alcance) vai variar com a velocidade do veículo para um alcance máximo de 300 m (~1000 ft). O mínimo alcance será 110 m (~367 ft). Por exemplo, veículos viajando a 96 km/h transmitirá a um nível de potência apropriado para alcançar aproximadamente 270 m (~880 ft) e veículos viajando a 40 km/h transmitirá a um nível de potência apropriado para alcançar aproximadamente 110 m (~367 ft). Todos os veículos capazes efectuarem este tipo de transmissão (possuírem OBU e com velocidade do veículo e posição válidos) vão transmitir estas mensagens e todos OBUs vão receber estas mensagens.

- Os veículos que recebem estas transmissões e possuem capacidade de processamento de aviso de colisão, computorizam a posição e probabilidade de colisão para todos os veículos que transmitem, a cada 100 ms.

- Uma precaução é enviada aos condutores quando existe a possibilidade de colisão se for computorizado um aviso de manobra que exceda .35 g ou a aceleração equivalente para as condições.

- Um aviso é enviado aos condutores quando existe a possibilidade de colisão se for computorizado um aviso de manobra que exceda .50 g ou a aceleração equivalente para as condições.

- Se for possível determinar que dois veículos se encontram em rota de intersecção, ambos usarão o alcance de transmissão do veículo mais rápido [10].

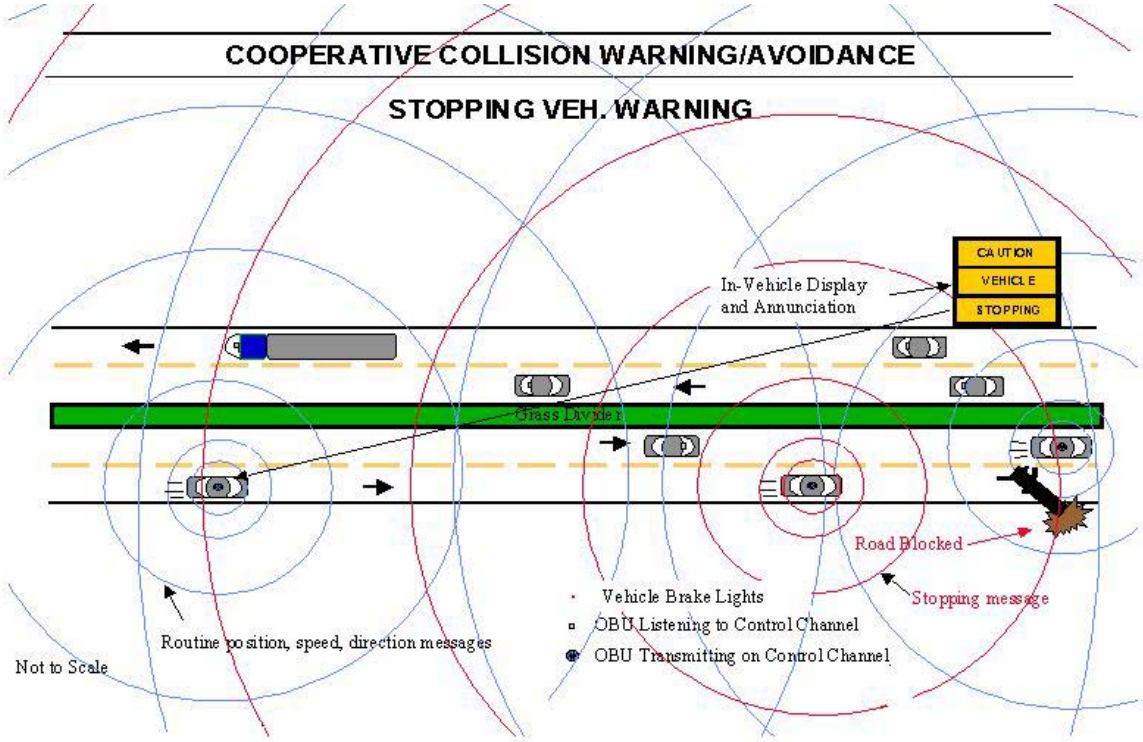

Figura 21: Aplicação veículo para veículo caso de aviso de paragem de veículo [10]

2.6 – VHDL

VHDL ou "**V**HSIC **H**ardware **D**escription **L**anguage" (Linguagem de descrição de hardware VHSIC "**V**ery **H**igh **S**peed **I**ntegrated **C**ircuits") é uma linguagem usada para facilitar o design (projecto/concepção) de circuitos digitais em FPGAs e ASICs [11].

A VHDL foi originalmente desenvolvida sob o comando do Departamento de Defesa dos Estados Unidos (DARPA), em meados da década de 80, para documentar o comportamento de ASICs que compunham os equipamentos vendidos às Forças Armadas americanas.

Isto quer dizer, que a linguagem VHDL, foi desenvolvida para substituir os complexos manuais que descreviam o funcionamento dos ASICs. Até aquele momento, a única metodologia largamente utilizada no projecto de circuitos era a criação através de diagramas esquemáticos. Os problemas com esta metodologia são, o facto de que o desenho tem menor portabilidade, são mais complexos para compreensão e são extremamente dependentes da ferramenta utilizada para produzi-los [11].

Uma vez que o projecto VHSIC era de alta prioridade militar e haviam dezenas de fornecedores envolvidos, o DoD estava preocupado principalmente

com as questões de portabilidade, documentação e compreensão dos projectos. Cada um destes fornecedores actuava desenvolvendo partes dos projectos ou mesmo fornecendo componentes que viriam a encaixar em outros sistemas maiores.

Desta forma o DoD optou por desenvolver uma linguagem que servisse como base para troca de informações sobre estes componentes e projectos. Uma linguagem que, independente do formato original do circuito, pudesse servir como uma descrição e documentação eficientes do circuito, possibilitando os mais diferentes fornecedores e participantes a entender o funcionamento das outras partes, padronizando a comunicação.

O desenvolvimento da VHDL serviu inicialmente aos propósitos de documentação do projecto VHSIC. Entretanto, nesta época procurava-se uma linguagem que facilitasse o projecto de um circuito, ou seja, a partir de uma descrição textual, um algoritmo, fosse desenvolvido o circuito, sem necessidade de especificar explicitamente as ligações entre componentes.

A VHDL presta-se adequadamente a tais propósitos, podendo ser utilizada para as tarefas de documentação, descrição, síntese, simulação, teste, verificação formal e ainda compilação de software, em alguns casos.

Após o sucesso inicial do uso da VHDL, a sua definição foi posta no domínio público, o que levou a ser padronizada pelo IEEE (Institute of Electrical and Electronic Engineers) em 1987. O facto de ser padronizada e de domínio público ampliou ainda mais a sua utilização, novas alterações foram propostas, como é natural num processo de aprimoramento e a linguagem sofreu uma revisão e um novo padrão mais actualizado, lançado em 1993. Actualmente ela continua a ser revista e deverá ser lançado um novo padrão em breve.

A VHDL, bem como outras linguagens, seguem um fluxo de projecto bem definido, composto de sete etapas, como apresenta a Figura 22:Especificação de Requisitos, Modelação, Síntese de Alto Nível, Mapeamento Tecnológico, Implementação e ou Fabricação, Testes e Simulação. O tempo e o custo de cada etapa dentro de um projecto são bastante variáveis, dependendo da tecnologia utilizada para implementar o sistema [11].

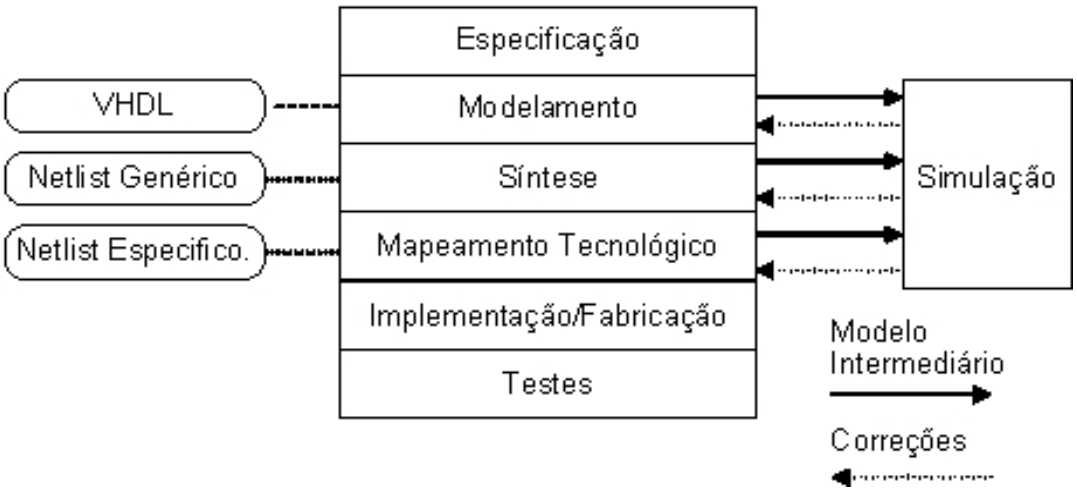

Figura 22: Ciclo de vida de um projecto [11]

Durante a etapa de Especificação de Requisitos, o projectista e o utilizador (em muitos casos podem ser a mesma pessoa), fazem um estudo e levantam todos os requisitos e características do sistema e definem o seu funcionamento. Características tais como, o atraso máximo permitido para as saídas, frequência máxima de operação, consumo de potência, custo, temperatura de operação, tensão de alimentação estão relacionadas com o fim de projectar um circuito que atenda a estes requisitos, que podem ser desejáveis ou necessários.

Não é necessário especificar todas estas características, isso sempre dependerá de cada projecto. Esta fase é de extrema importância porque uma vez compreendido correctamente o funcionamento do sistema, evita-se a ocorrência de erros futuros. A cada unidade de tempo avançada no ciclo de projecto, maior é o custo de correcção de um erro e maior a dificuldade em encontrá-lo, ou seja, além do prejuízo ser maior, maior também é a probabilidade de que o erro passe despercebido e seja incluído na produção do sistema.

Na etapa de modelação é quando o projecto propriamente dito é iniciado. Baseado nas especificações da etapa inicial, o projectista irá escrever os modelos que representam o circuito. É de fundamental importância que o projectista tenha já um prévio conhecimento das ferramentas de software que utilizará no projecto e da tecnologia que irá utilizar, a fim de conduzir a modelação, a fim de obter os melhores resultados. Outras características importantes a serem incluídas nos modelos são: sempre que possível, de maneira a não afectar o desempenho e a portabilidade, escrever modelos que sigam o padrão estabelecido na linguagem, e não, as extensões oferecidas pelos desenvolvedores das ferramentas de síntese, seguir um padrão de escrita de código uniforme para todos os modelos do

projecto, documentar adequadamente os modelos, incluindo nome do autor, datas de manutenção, e comentários e explicações relevantes.

A Síntese de Alto Nível está para o hardware assim como a compilação está para o software. Na síntese, o modelo descrito será convertido para estruturas de dados representando as conexões, blocos, componentes e portas lógicas. Esta etapa é automática e dependente da ferramenta de software utilizada. Em geral, certos cuidados podem ser tomados durante a modelação com o fim de direccionar o algoritmo da síntese para que obtenham os melhores resultados para o circuito. Durante a síntese são pré-avaliados os requisitos do sistema, a fim de indicar se os circuitos irão atendê-los adequadamente. Após a síntese ainda não está definido o circuito a ser implementado, a especificação intermediária que é resultante ainda é bastante genérica e pode ser direccionada para uma de muitas possibilidades de tecnologias de implementação. Somente após o Mapeamento Tecnológico é que o circuito já está definido dentro da tecnologia em que será implementado. Fazendo uma analogia com o software, essa etapa corresponderia à geração de código executável que ocorre ao final da compilação de um código fonte. Só é possível entender essa etapa adequadamente conhecendo-se as diferentes tecnologias disponíveis no mercado, como full custom, gate array, FPGAs, entre outros.

O projectista pouco consegue influir no mapeamento, especificando apenas os parâmetros de optimização desejados. Na etapa de implementação/fabricação não há muito a ser explicado, neste momento são criados os primeiros protótipos, avaliadas as condições finais, detalhes de produção entre outros detalhes de implementação final. Em seguida à fabricação, os circuitos são testados para que possam ser entregues ao utilizador com garantia de isenção de falhas.

A Simulação é uma etapa auxiliar, mas de grande relevância no ciclo de vida do projecto. Simular consiste em avaliar o comportamento do circuito e validar o modelo produzido até aquele momento. Durante a simulação, são apresentadas amostras de entradas possíveis ao modelo do circuito, e os valores das saídas, memórias e nós internos do circuito são analisados a fim de comparar com o esperado na especificação.

A Simulação gera uma realimentação para os processos de modelação, síntese e mapeamento, evitando a propagação de erros para etapas posteriores. Muitos dos problemas encontrados na simulação não estão necessariamente ligados a erros no projecto, mas ao não preenchimento dos requisitos necessários, principalmente no que se refere aos tempos do circuito (atraso, setup/hold, frequência de operação).

Na simulação, um recurso muito interessante a ser utilizado são os testbenches. Estes modelos não geram circuitos, servindo apenas para a

simulação, o testebench nada mais é que um algoritmo responsável por gerar automaticamente os sinais que simularão as entradas do modelo que será simulado. Dentro do testebench podem também estar presentes rotinas que capturem os valores de saída do circuito simulado e gerem apenas resultados de comparação dizendo se o circuito está funcionando correctamente ou não e quais aspectos que possuem problemas [11].

2.7 – FPGA´S, ALTERA e QUARTUS II

2.7.1 – FPGA´S

Com a evolução da microelectrónica nos últimos anos, deparamo-nos com uma diversidade de dispositivos electrónicos mais rápidos, com maior capacidade para armazenamento de informações, menor consumo de energia e custos cada vez mais baixos, colocando os projectistas de computadores num constante desafio: o balanço entre velocidade de operação dos sistemas versus a generalidade de operação desses sistemas.

Da mesma forma que os projectistas necessitam de decidir sobre versatilidade e velocidade, eles também precisam confrontar-se com questões de custo. Se considerarmos por exemplo, um sistema implementado através de um ASIC, que resolverá um problema específico, nada poderá ser modificado após o projecto ter sido concluído. Mesmo que esse projecto seja reutilizado, o custo dessa reutilização também seria amortizado com a venda das primeiras unidades.

Um novo desenvolvimento em circuitos integrados, oferece uma terceira opção com grande capacidade e velocidade, que são os "Field Programmable Gate Array" ou FPGAs, circuitos de hardware que podem ser modificados praticamente em qualquer momento durante o uso. Os dispositivos FPGAs consistem de um array de blocos lógicos configuráveis, que implementam funções lógicas AND, NAND, OR, NOR e XOR.

Hoje em dia, na maioria dos hardwares, as suas funções lógicas são fixas e não podem ser modificadas. No entanto, os FPGAs, podem ser alterados, tanto nas funções lógicas como nos blocos lógicos, além das conexões entre esses blocos, simplesmente enviando sinais de configuração para os "chips". Os blocos lógicos num FPGA podem ser rescritos e reprogramados repetidamente, muito depois do "chip" ter sido produzido na fábrica [36].

Um FPGA é um dispositivo semicondutor que é largamente utilizado para o processamento de informações digitais. Foi criado pela Xilinx Inc., e teve o seu lançamento no ano de 1983 como um dispositivo que poderia ser programado de acordo com as aplicações do utilizador (programador). O FPGA é composto basicamente por três tipos de componentes: blocos de entrada e saída (IOB), blocos lógicos configuráveis (CLB) e chaves de interconexão (Switch Matrix).

Os blocos lógicos são dispostos de forma bidimensional, as chaves de interconexão são dispostas em formas de trilhas verticais e horizontais entre as linhas e as colunas dos blocos lógicos.

- CLB (Configuration Logical Blocks): Circuitos idênticos, construídos pela reunião de flip-flops (entre 2 e 4) e a utilização de lógica combinatória. Utilizando os CLBs, o utilizador pode construir elementos funcionais lógicos.

- IOB (Input/Output Block): São circuitos responsáveis pela interface das saídas provenientes das saídas das combinações de CLBs. São basicamente *buffers*, que funcionarão como um pino bidireccional de entrada e saída do FPGA.

- Switch Matrix (chaves de interconexões): Trilhas utilizadas para conectar os CLBS e IOBS. O terceiro grupo é composto pelas interconexões. Os recursos de interconexões possuem trilhas para conectar as entradas e saídas dos CLBs e IOBs para as redes apropriadas. Geralmente, a configuração é estabelecida por programação interna das células de memória estática, que determinam funções lógicas e conexões internas implementadas no FPGA entre os CLBs e os IOBs. O processo de escolha das interconexões é chamado de roteamento [12].

Actualmente no mercado podemos encontrar três tipos de FPGAs, onde cada um terá melhor desempenho dependendo da aplicação para a qual o mesmo será utilizado. Os três tipos são:

- RAM Estática: FPGA na qual as suas conexões entre as portas são feitas entre blocos lógicos por meio de portas de transmissão ou multiplexadores controladas por células SRAM. Tem como vantagem a possibilidade de ser rapidamente configurada, porém exige hardware externo auxiliar que deve ser montado junto com os blocos lógicos.
- Transístores de Passagem: Esta é uma opção mais barata que a opção de RAM estática, composta por uma grande concentração de transístores que são configurados em modo de corte ou modo de condução.
- EPROM/EEPROM: Baseada na tecnologia de criação de memórias EPROM/EEPROM. Sua principal vantagem é permitir a reprogramação sem que se precise armazenar a configuração externa [12].

Na figura 23 encontra-se uma FPGA da família Stratix, que é o modelo do kit usado no Instituto de Telecomunicações para o desenvolvimento e teste do projecto.

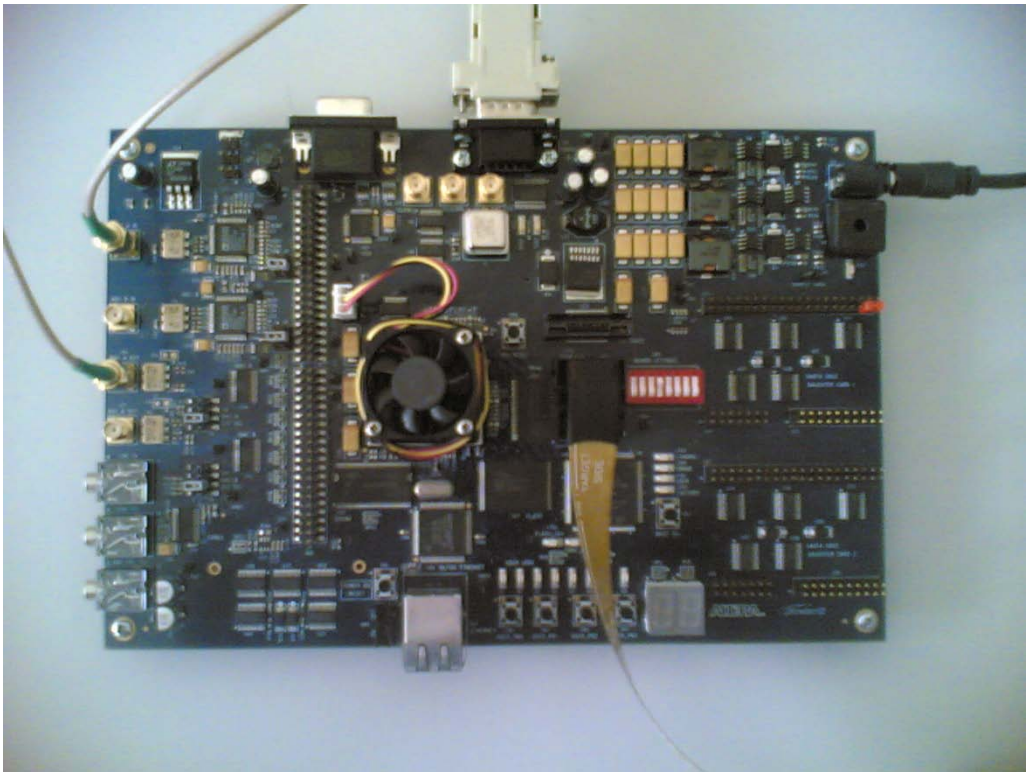

Figura 23: FPGA Altera (Kit do Instituto de Telecomunicações)

2.7.2 – Altera

A Altera é uma empresa fundada em 1983, sediada na zona de Silicon Valey, nos EUA, é uma das maiores empresas mundiais em soluções de lógica reconfigurável. Hoje, mais de 2600 empregados em 19 países estão desenvolvendo soluções lógicas mais evoluídas. Abrangendo uma gama de conceitos, desde, consumo de potência a desempenho e custo, produzem-se soluções para clientes de uma grande variedade de indústrias, incluindo automóvel, radiodifusão, computacional e armazenamento, médica, exército, testes e medidas, telegrafia sem fios. Para complemento dos dispositivos, a Altera desenvolve ferramentas de software completamente integradas, processadores embebidos, IP cores, designs de referência e uma variedade de equipamentos de desenvolvimento [13].

Figura 24: Empresa Altera em San Jose (Silicon Valey) [14]

Os principais produtos da Altera são as famílias de FPGAs: Cyclone, ArriaGX e Stratix, a família MAX de CPLDs (Complex Programmable Logic Devices), a família Hardcopy dos ASICs (Structured ASIC) e o software Quartus II [14].

2.7.3 Quartus II

O software Quartus II dispõe de um ambiente de desenho completo para systems-on-a-programmable-chip (SOPC). Sem ter em consideração se o utilizador usa um computador pessoal ou um sistema UNIX ou uma estação de trabalho Linux, o Quartus II garante uma entrada fácil no desenho, processamento rápido, e uma programação rápida e eficiente.

O Quartus II é um pacote integrado, de arquitectura independente, para desenho de lógica com dispositivos lógicos programáveis da Altera, incluindo os dispositivos ACEX 1K, APEX 20K, APEX 20KC, APEX 20KE, APEX II, ARM, Cyclone, FLEX 6000, FLEX 10K, FLEX 10KA, FLEX 10KE, MAX® 3000A, MAX 7000AE, MAX 7000B, MAX 7000S, Mercury, Stratix, and Stratix™ GX.

O Quartus II oferece larga capacidade de desenho lógico, como, esquemáticos de desenho, diagramas de blocos, AHDL, VHDL, e Verilog HDL, edição, simulação funcional e temporal, análise temporal, compilação e projectos de software, programação de dispositivos e sua verificação. O Quartus II também

lê ficheiros standards EDIF netlist, ficheiros VHDL netlist, ficheiros Verilog HDL netlist, e gera ficheiros VHDL e Verilog HDL netlist, incluindo ficheiros compatíveis VITAL, para uma interface conveniente para outras ferramentas de indústria standard EDA [9].

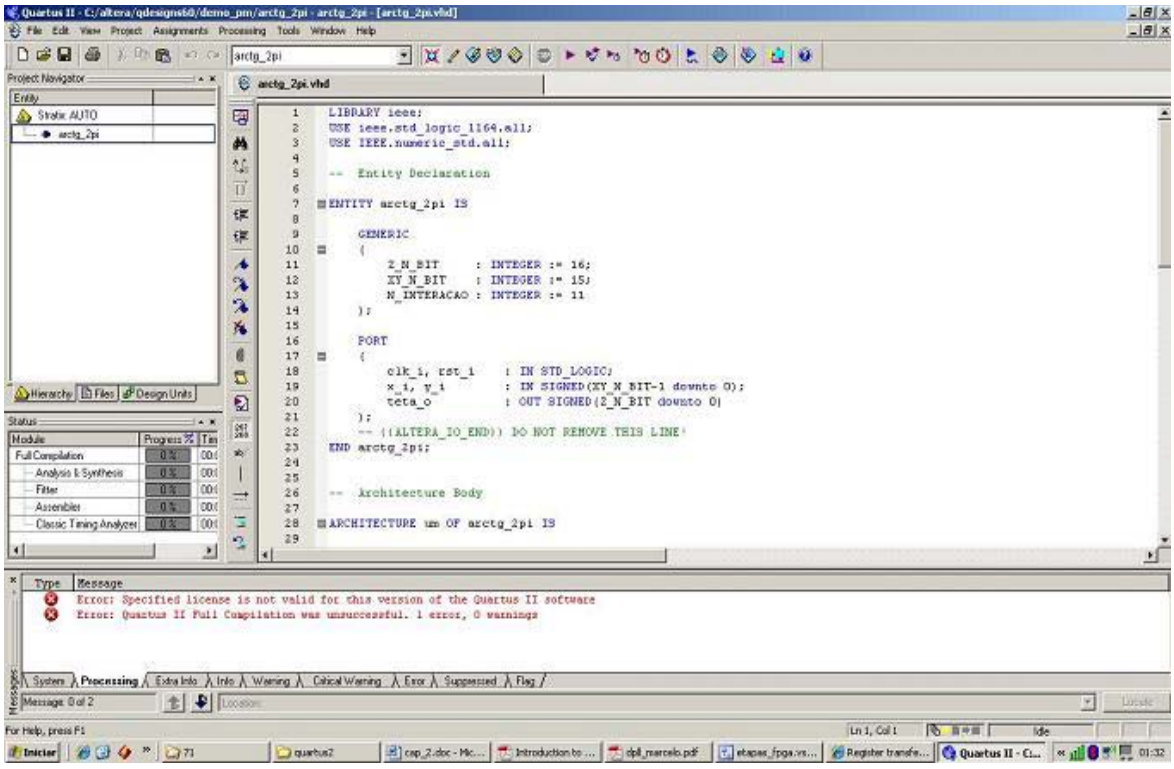

Figura 25: Janela de trabalho do Quartus II no ambiente de edição de código HDL [15]

Capítulo III – Conceptualização do sistema

3.1 – Conceptualização do bloco extensão cíclica

O sistema a ser implementado em linguagem VHDL, no software Quartus II Web edition v.9.0 da Altera trata-se do bloco de extensão cíclica pertencente ao transmissor, o qual já foi referido no capítulo 2, trata-se do bloco que introduz um prefixo aos blocos OFDM.

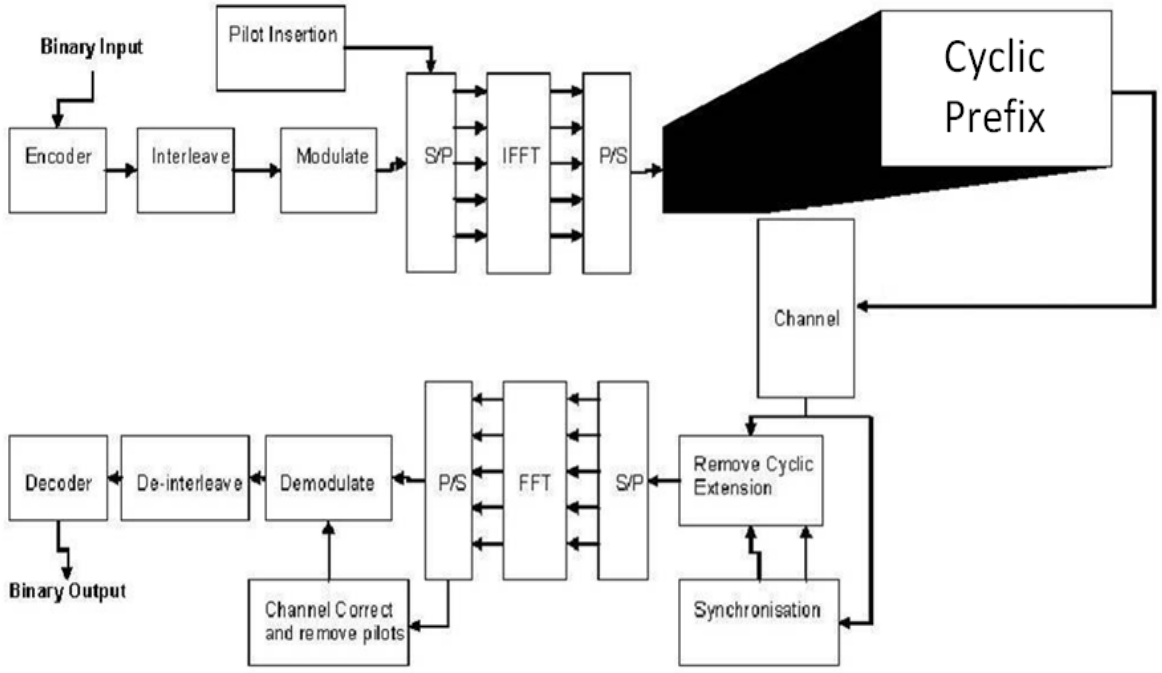

Figura 26: Bloco pertencente ao transceiver a ser implementado [6]

Para a conceptualização deste bloco em VHDL, surgiram uma série de ideias para levar a cabo esta tarefa da melhor maneira possível. Deste modo, e de acordo com a ideia mais simplista, o bloco foi conceptualizado para quatro grandes partes: a parte selectora, a parte de controlo dos Fifos, os Fifos, e a parte de atraso de um dos sinais com a soma dos sinais no final. Como foi visto anteriormente no capítulo II na secção 2.4.2.6, então o bloco deverá introduzir o prefixo como se pode ver um exemplo ilustrativo na figura 27.

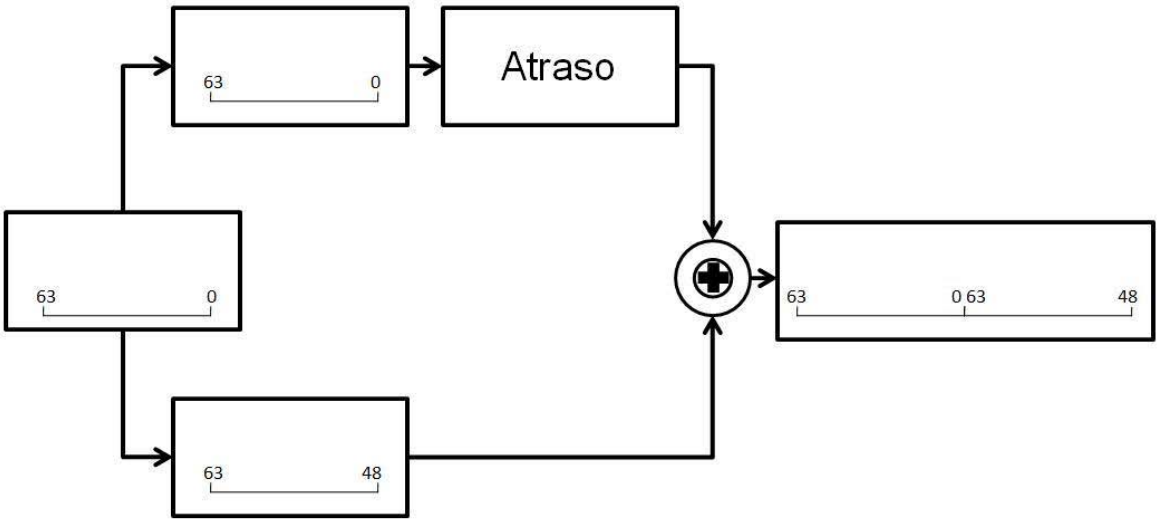

Figura 27: Esquema ilustrativo da operação de adição do prefixo

3.2 – Parte Selectora

O bloco selector faz a distinção entre o bloco de 64 bits e a parte desse bloco correspondente ao prefixo. Esta distinção é feita e produzida nos sinais Sel1 e Sel2, sendo o Sel1 activo ‘1’ durante o próprio bloco de 64 bits (ou seja, sempre que existirem dados válidos) e o Sel2 está activo ‘1’ durante os últimos 16 bits de cada bloco de 64 bits.

Figura 28: Parte selectora do bloco

3.3 – Parte de controlo dos Fifos

Nesta parte é feito o controlo dos Fifos ao nível da escrita e leitura de dados. Cada controlador recebe a informação correspondente ao sinal de selecção feito no bloco selector. De acordo com a informação recebida, faz a ordenação da escrita e leitura do Fifo correspondente.

Figura 29: Parte controladora do Fifo

Figura 30: Parte controladora do Fifo2

3.4 – Fifos

Nesta parte do sistema é onde é armazenada a informação e posteriormente libertada de acordo com a parte controladora. No Fifo são armazenados os blocos de 64 bits e no Fifo2 são armazenados os prefixos de16 bits.

Figura 31: Fifo

Figura 32: Fifo2

3.5 – ShiftRegister e Or

No final do sistema surge um ShiftRegister cuja única função é atrasar os dados que são libertados do Fifo. Na saída do sistema a função Or realiza a junção dos dados libertados dos dois Fifos, perfazendo o sinal de saída desejado.

Figura 33: ShiftRegister

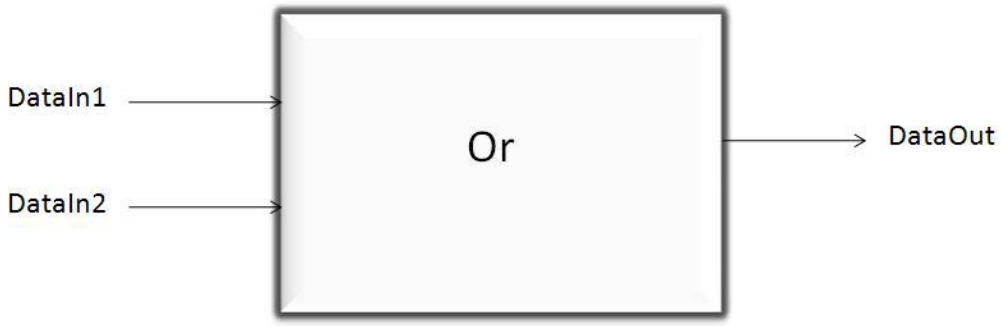

Figura 34: Or

Articulando todos estes sub-blocos teremos então a solução final encontrada, indo de encontro ao objectivo final pretendido.

Capítulo IV – Implementação e resultados do sistema

4.1 – Vista geral do sistema

Neste capítulo mostra-se a implementação do sistema no Quartus II, no qual são visíveis todos os sub-blocos constituintes do sistema, bem como as entradas e saídas dos mesmos. Estes sub-blocos são mostrados desde a entrada até à saída por ordem decorrente do processo implementado, sendo isso visível na figura 37 e mais pormenorizadamente por partes nas figuras 35 e 36.

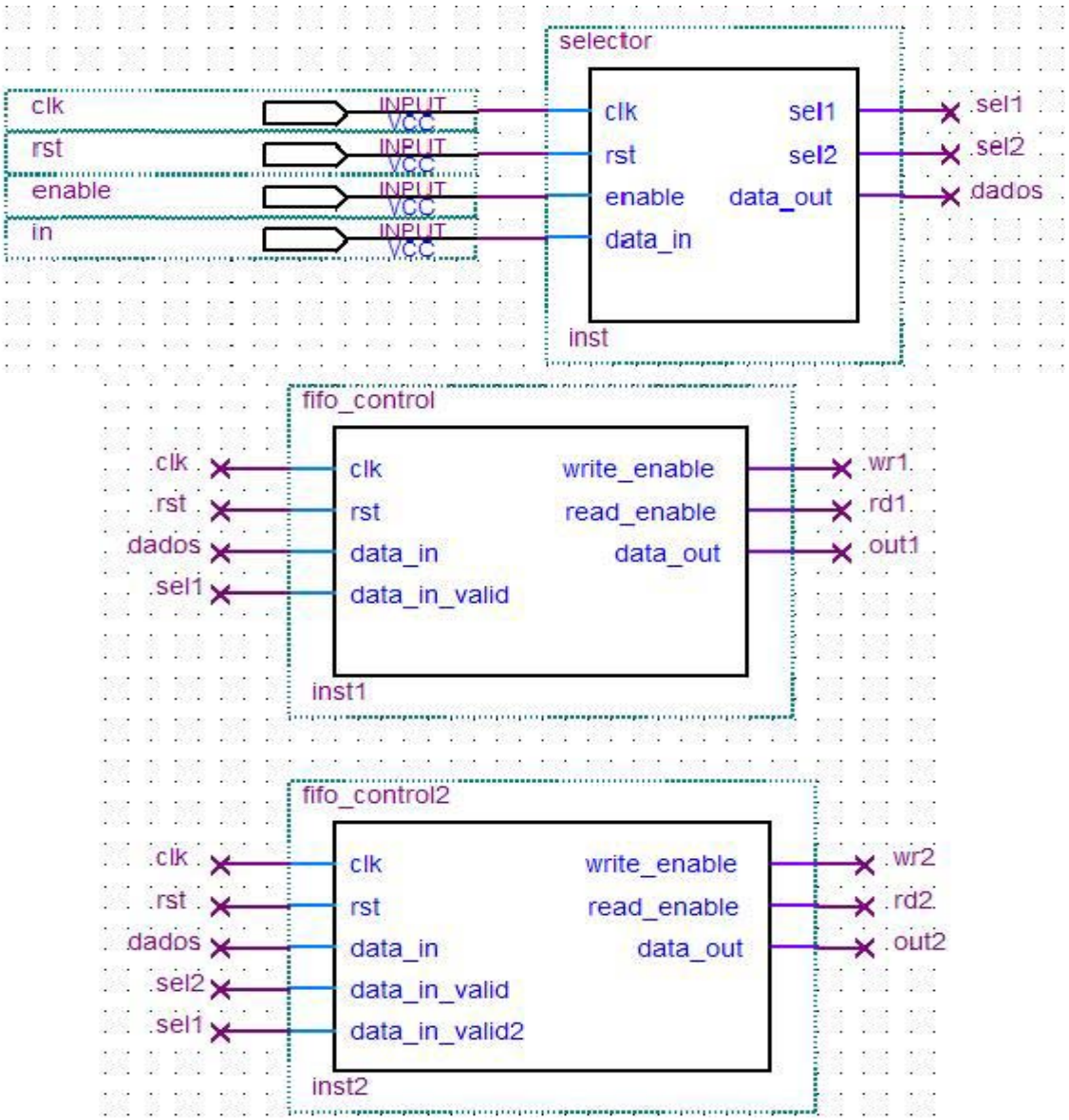

Figura 35: Vista geral do sistema (selector, fifo_control e fifo_control2) implementado em ambiente block design no Altera Quartus II

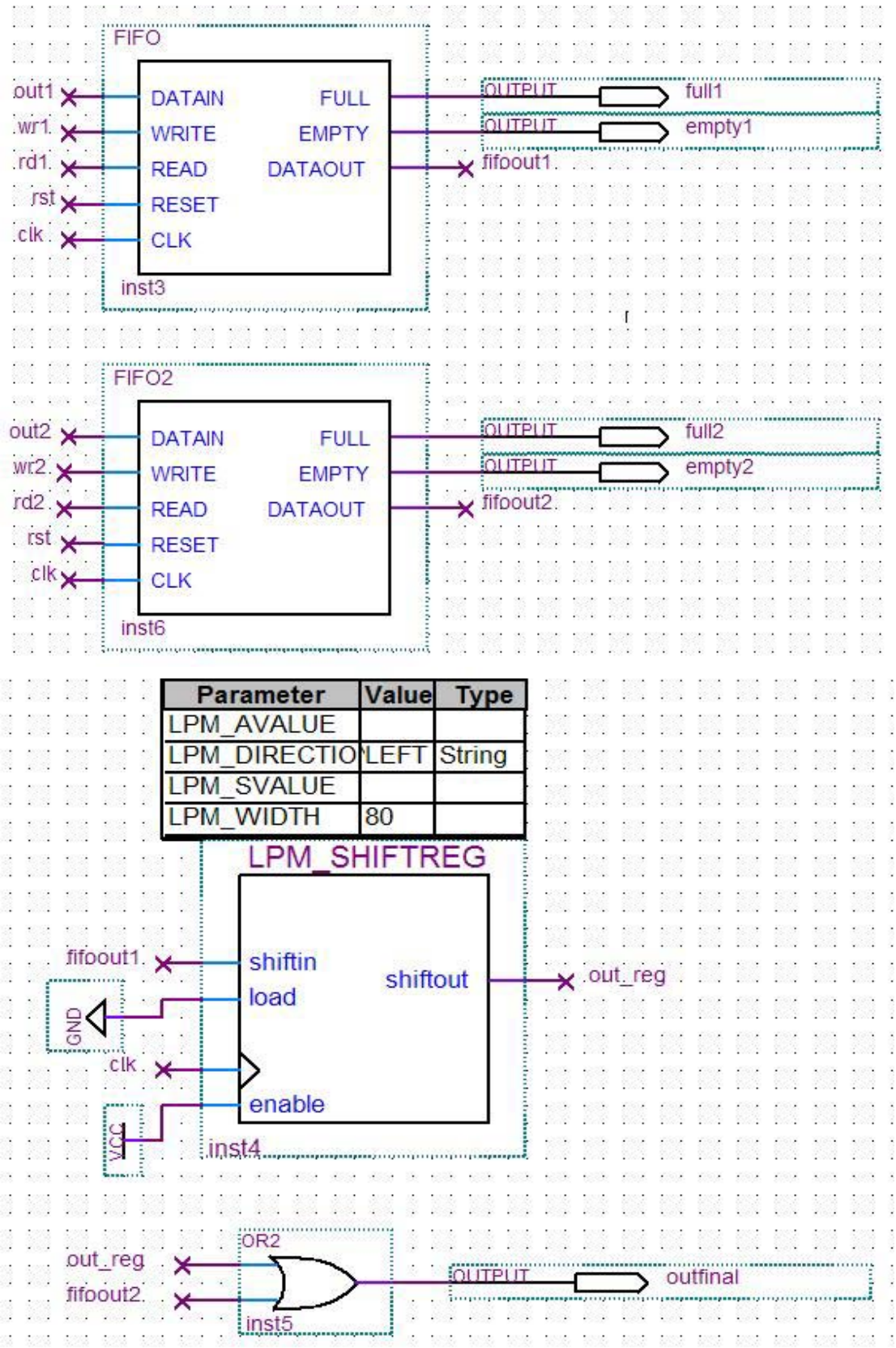

Figura 36: Vista geral do sistema (fifo, fifo2, lpm_shiftreg e or) implementado em ambiente block design no Altera Quartus II

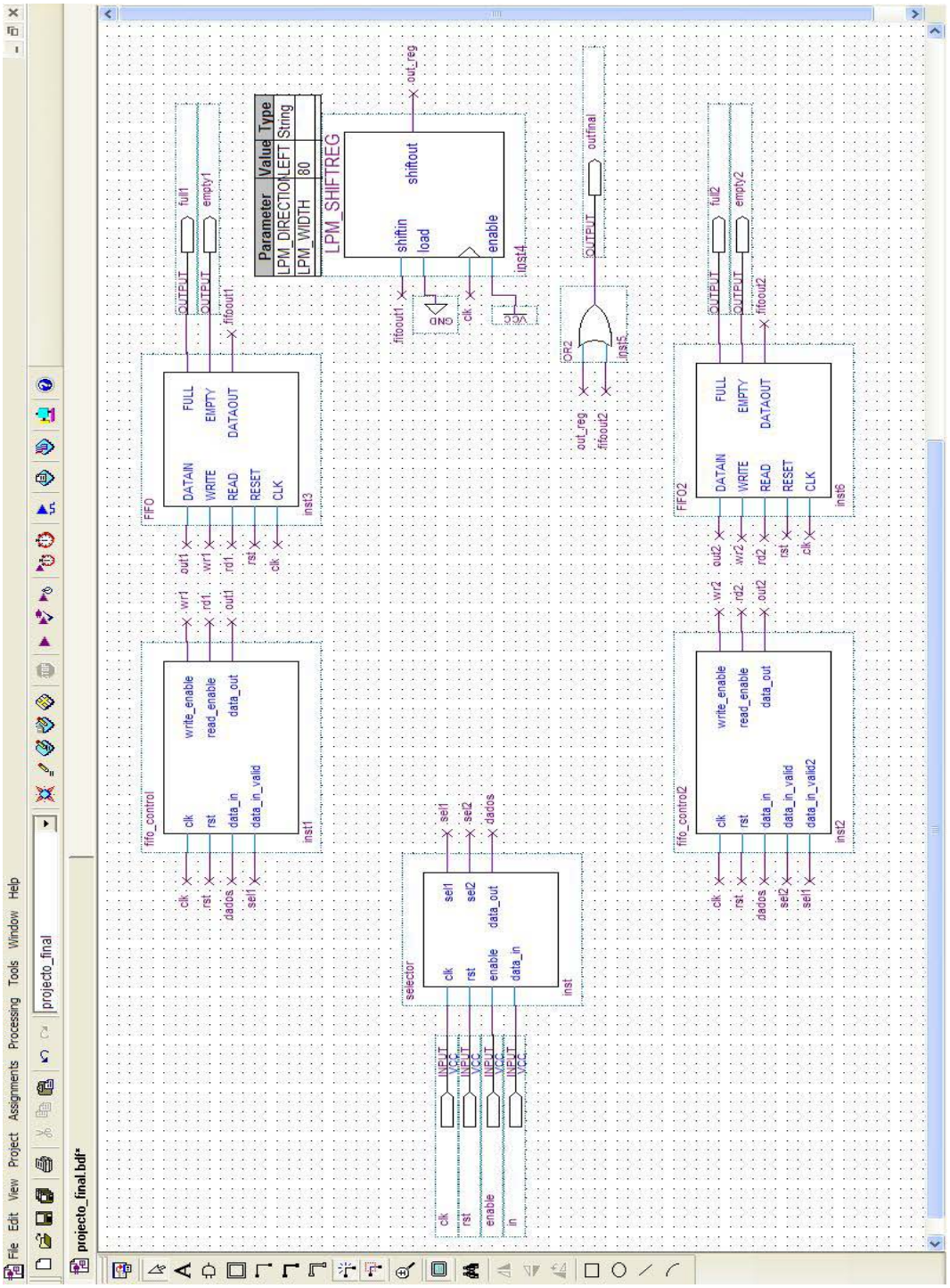

Figura 37: Vista geral do sistema implementado em ambiente block design no Altera Quartus II

Como pode ser visto na figura 38, a transmissão de um único bloco de 64 bits processa-se de acordo com o pretendido, já que, na entrada in, deram entrada 64 bits validados pelo sinal de enable, e na saída outfinal encontram-se 80 bits, divididos por 16 bits iniciais que são uma cópia dos 16 últimos bits recebidos na entrada, constituindo o prefixo, mais os 64 bits originais da entrada.

Nas figuras 39 e 40 visualiza-se a chegada na entrada in de dois blocos de 64 bits cada, um determinado tempo depois do bloco anterior ter sido recebido, sendo que na saída outfinal, surgem 160 bits correspondentes a, 16 bits de prefixo do primeiro bloco, 64 bits do primeiro bloco, 16 bits de prefixo do segundo bloco e finalmente 64 bits do segundo bloco.

4.1.1 – Resultados da simulação funcional do sistema

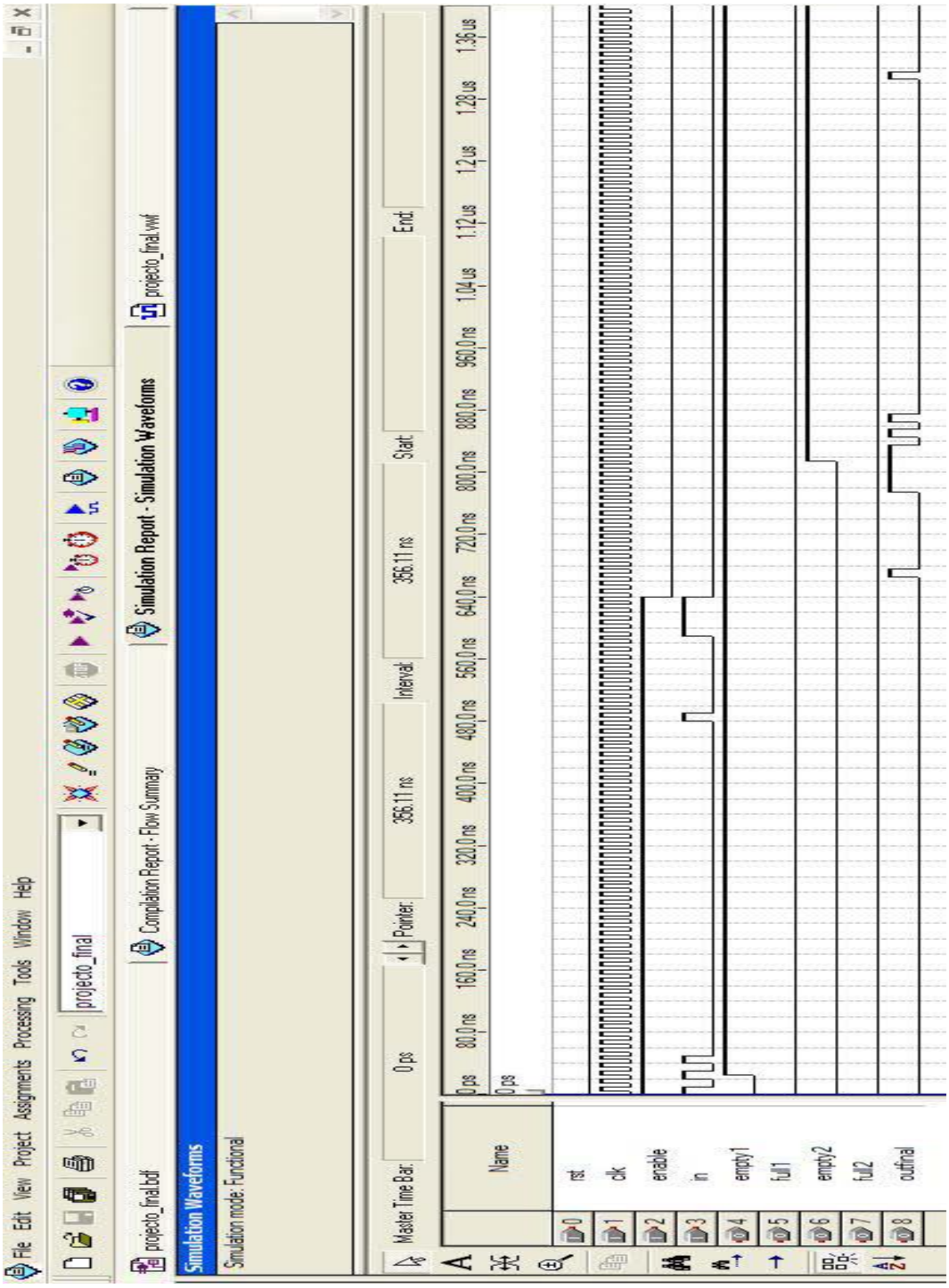

Figura 38: Resultado funcional obtido do sistema no intervalo [0;1,4] µs

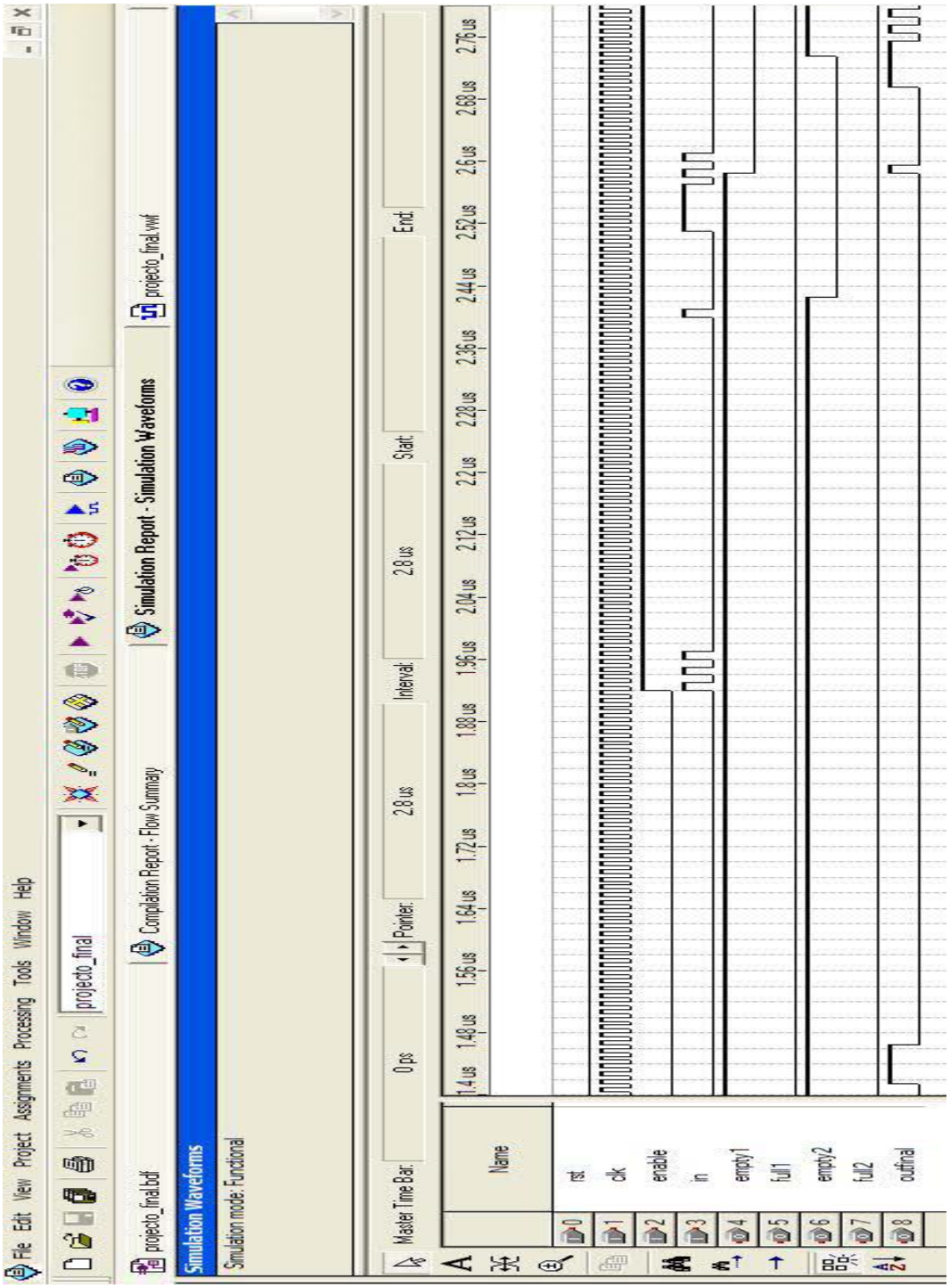

Figura 39: Resultado funcional obtido do sistema no intervalo [1,4;2,8] µs

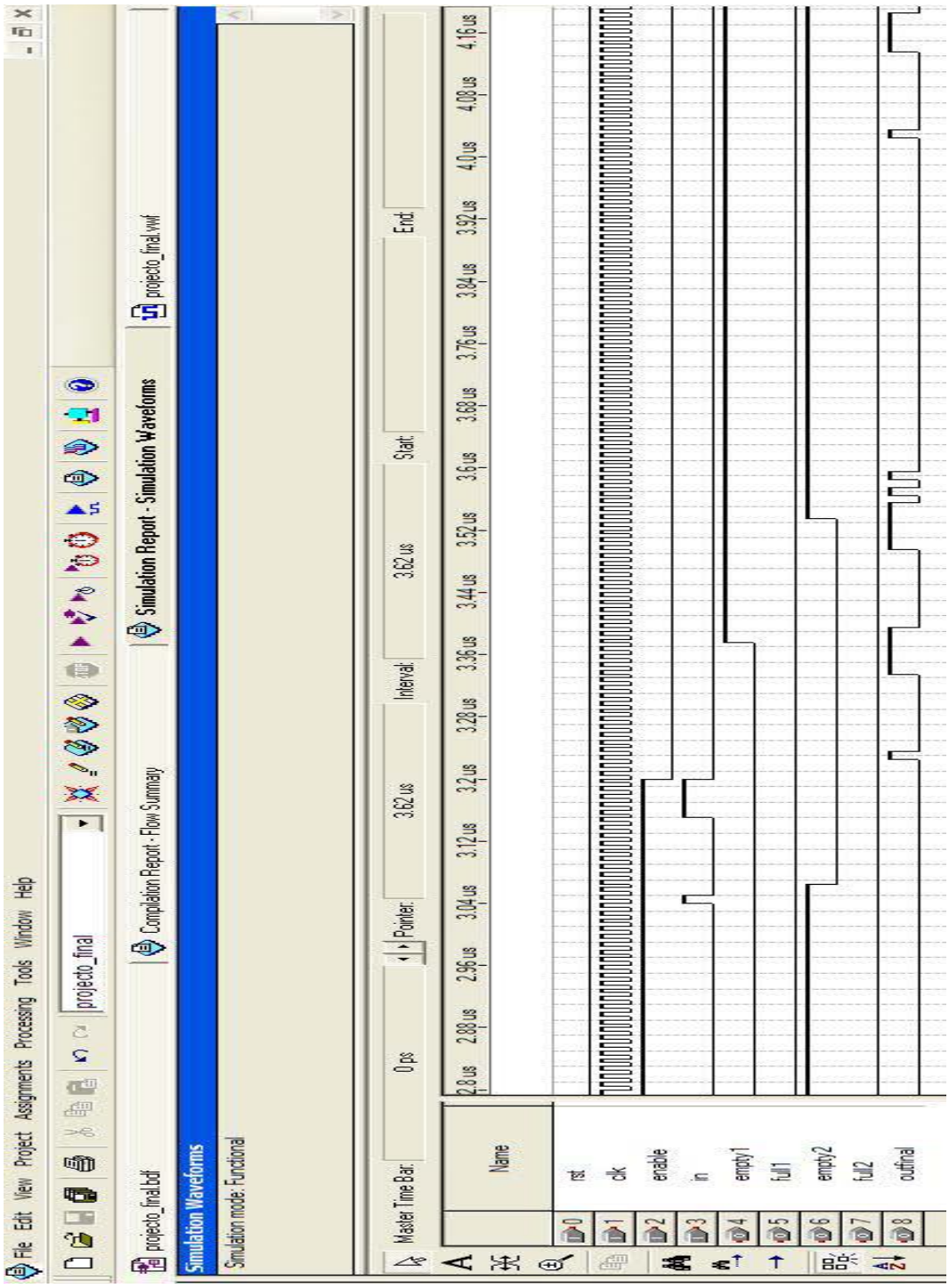

Figura 40: Resultado funcional obtido do sistema no intervalo [2,8;4,2] µs

4.2 – Selector

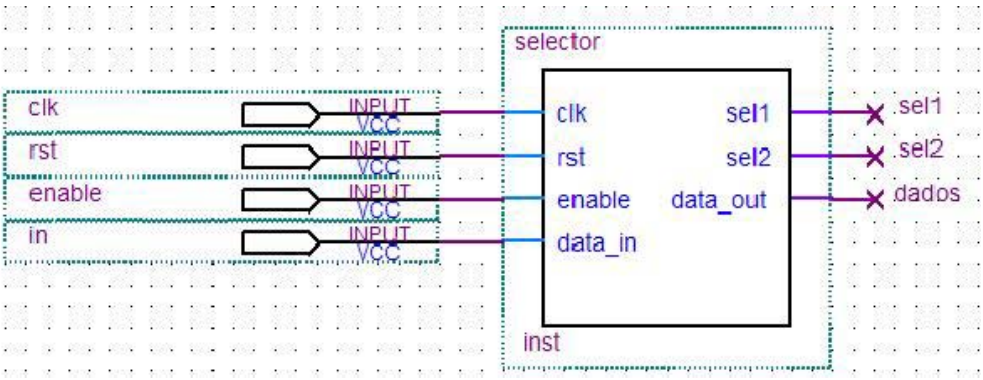

Figura 41: Selector implementado no Quartus II

4.2.1 – Características do selector

Tabela 7: Características do bloco Selector

Portas de entrada	clk
	rst
	enable
	data_in
Portas de saída	sel1
	sel2
	data_out
Sinais	contador

4.2.2 Descrição das operações do selector

- Sempre que reset esteja activo (‘1’), as saídas e sinais são remetidos a zero.
- Por cada transição ascendente de relógio são efectuadas as seguintes operações.
- É verificada a entrada de validação enable, ou seja, se está activa (‘1’), senão todas as saídas são colocadas a zero e é verificado se o contador tem o valor 64, e se sim, é remetido a zero.
- A saída sel1 é colocada a 1, os dados de entrada data_in são colocados na saída data_out e é incrementado o sinal contador.
- Se o sinal contador estiver no intervalo de valores de 49 a 64, é colocada a saída sel2 a ‘1’, senão é a ‘0’.
- O sinal contador atingindo o valor 64, é reposto ao valor 1.

Na figura 42, está patente de novo a chegada de um bloco de 64 bits na entrada in, sendo que o sinal sel1 fica activo ‘1’ durante todo o bloco, ou seja, durante 64 bits, enquanto que o sinal sel2 apenas fica activo ‘1’, quando o contador se encontra compreendido entre os valores 49 e 64 inclusive, pois são os últimos 16 bits desse bloco. No final os sinais sel1 e sel2 voltam ao estado inactivo ‘0’, enquanto o contador é reposto a zero.

Na figura 43 é mostrado um trecho temporal em que estão dois blocos na entrada in, embora só seja visualizado o fim do primeiro e o segundo integralmente, mas dando a ver, que o sinal sel1 está activo ‘1’ durante os 128 bits resultantes de dois blocos de 64 bits cada e o sinal sel2 apenas está activo ‘1’ quando o contador se encontra entre 49 e 64 de cada bloco, estando inactivo ‘0’ no restante tempo. Quando o contador atinge o valor 64 e ainda existem mais dados então é remetido ao valor 1 para iniciar uma nova contagem de 64 bits relativos a um novo bloco e quando os dados deixam de ser válidos e termina a transmissão então toma o valor zero. No anexo A encontra-se o código VHDL do bloco selector.

4.2.3 – Resultados da simulação funcional do bloco selector

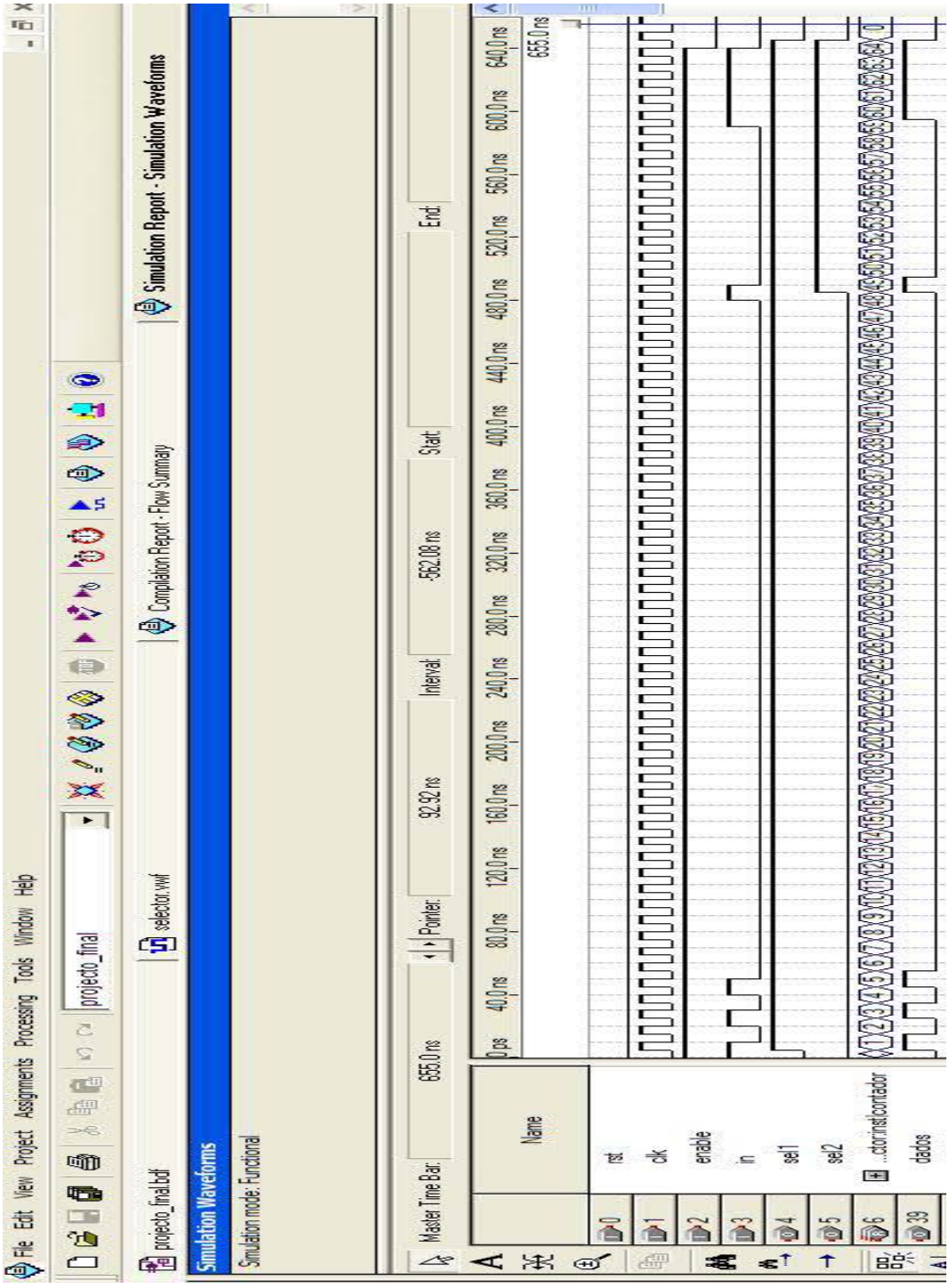

Figura 42: Resultados funcionais do bloco selector no intervalo [0;0,65] µs

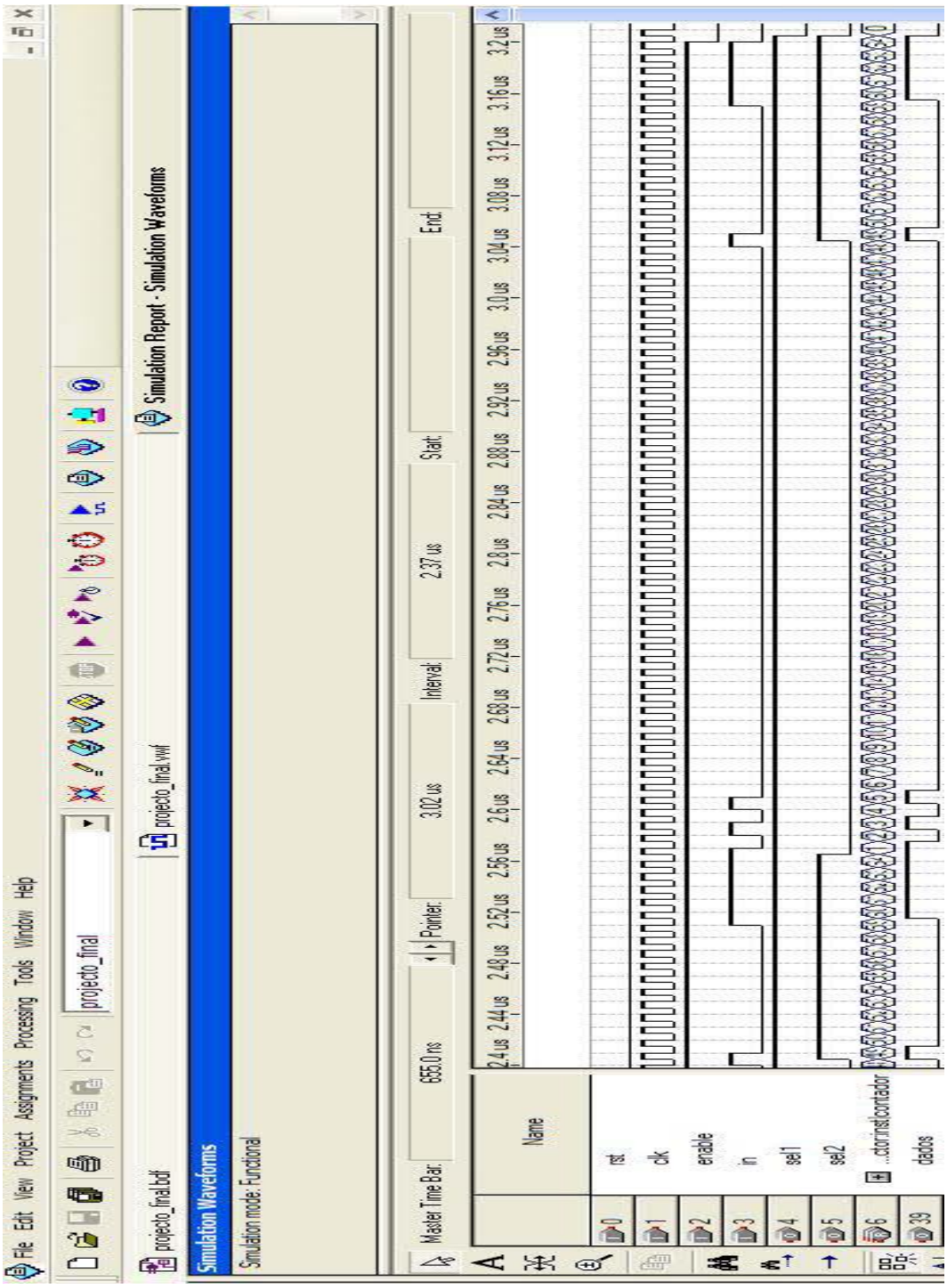

Figura 43: Resultados funcionais do bloco selector no intervalo [2,4;3,2] µs

4.3 – Fifo_control

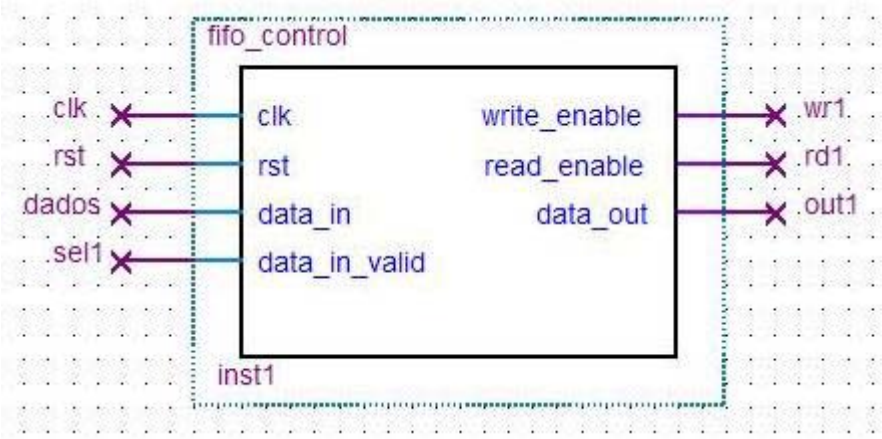

Figura 44: fifo_control implementado no Quartus II

4.3.1 – Características do fifo_control

Tabela 8: Características do bloco fifo_control

Portas de entrada	clk
	rst
	data_in
	data_in_valid
Portas de saída	write_enable
	read_enable
	data_out
Sinais	s_contador
	s_contador_in
	s_contador_out

4.3.2 – Descrição das operações do fifo_control

- Sempre que reset esteja activo (‘1’), os sinais são remetidos a zero e as saídas read_enable e write_enable também são remetidas a zero.
- Por cada transição ascendente de relógio são efectuadas as seguintes operações.
- É verificada a entrada de validação data_in_valid, ou seja, se está activa (‘1’), senão, as saídas write_enable e data_out são ‘0’ e o sinal s_contador é incrementado.

- Se o valor do sinal s_contador estiver compreendido entre 64 e 80 a saída read_enable é posta a ‘0’, senão, é posta a ‘1’, o sinal s_contador_out é incrementado e o valor do sinal s_contador é remetido a ‘1’.
- Se os valores dos sinais s_contador_in e s_contador_out forem iguais então todos os sinais e a saída read_enable são remetidos a ‘0’.
- Se a entrada de validação data_in_valid está activa (‘1’), então, os sinais s_contador e s_contador_in são incrementados, a saída write_enable é remetida a ‘1’ e a saída data_out toma os valores de data_in.
- Se o valor do sinal s_contador for menor que 64 então a saída read_enable é remetida a ‘1’ e o sinal s_contador_out é incrementado.
- Se o valor do sinal s_contador estiver compreendido entre 64 e 80, então, a saída read_enable é ‘0’, senão, é ‘1’, o sinal s_contador_out é incrementado e o sinal s_contador é remetido a ‘1’.

Na figura 45 é visível a transmissão de apenas um bloco de 64 bits de dados na entrada in sendo que os sinais de read1 e write1 estão activos ‘1’, o que significa que os dados estão a ser escritos e lidos em simultâneo. Os sinais contadores, s_contador_in e s_contador_out estão a ser incrementados pois os dados estão a ser escritos (write1) e lidos (read1) ao mesmo tempo. No final quando a transmissão deixa de ser válida (sinal enable fica inactivo ‘0’), os sinais read1 e write1 ficam inactivos ‘0’ e os contadores são inicializados a zero.

Nas figuras 46 e 47 mostram-se dois blocos de 64 bits cada na entrada in sendo que o sinal write1 está sempre activo ‘1’ durante esses 128 bits, porque obviamente os dados têm todos que ser escritos à medida que vão chegando na entrada in. O sinal read1 está activo ‘1’ durante 64 bits, sendo que depois está inactivo ‘0’ durante 16 bits, tal como o pretendido. É de notar que durante estes 16 bits em que o sinal read1 está inactivo o contador s_contador_out está parado com o mesmo valor, porque durante este período não foram libertados quaisquer dados.

O contador s_contador_in está sempre a ser incrementado porque o sinal write1 está sempre activo que é motivo de estarem a chegar dados à entrada in. No final os sinais read1 e write1 ficam inactivos ‘0’ pois os dados entretanto deixaram de ser válidos e quando o contador s_contador_out atingir o mesmo valor de s_contador_in, são remetidos ao valor zero, o que significa que todos os dados que foram escritos também já foram lidos e estão prontos a receber uma nova transmissão. No anexo B encontra-se o código VHDL do bloco fifo_control.

4.3.3 – Resultados da simulação funcional do fifo_control

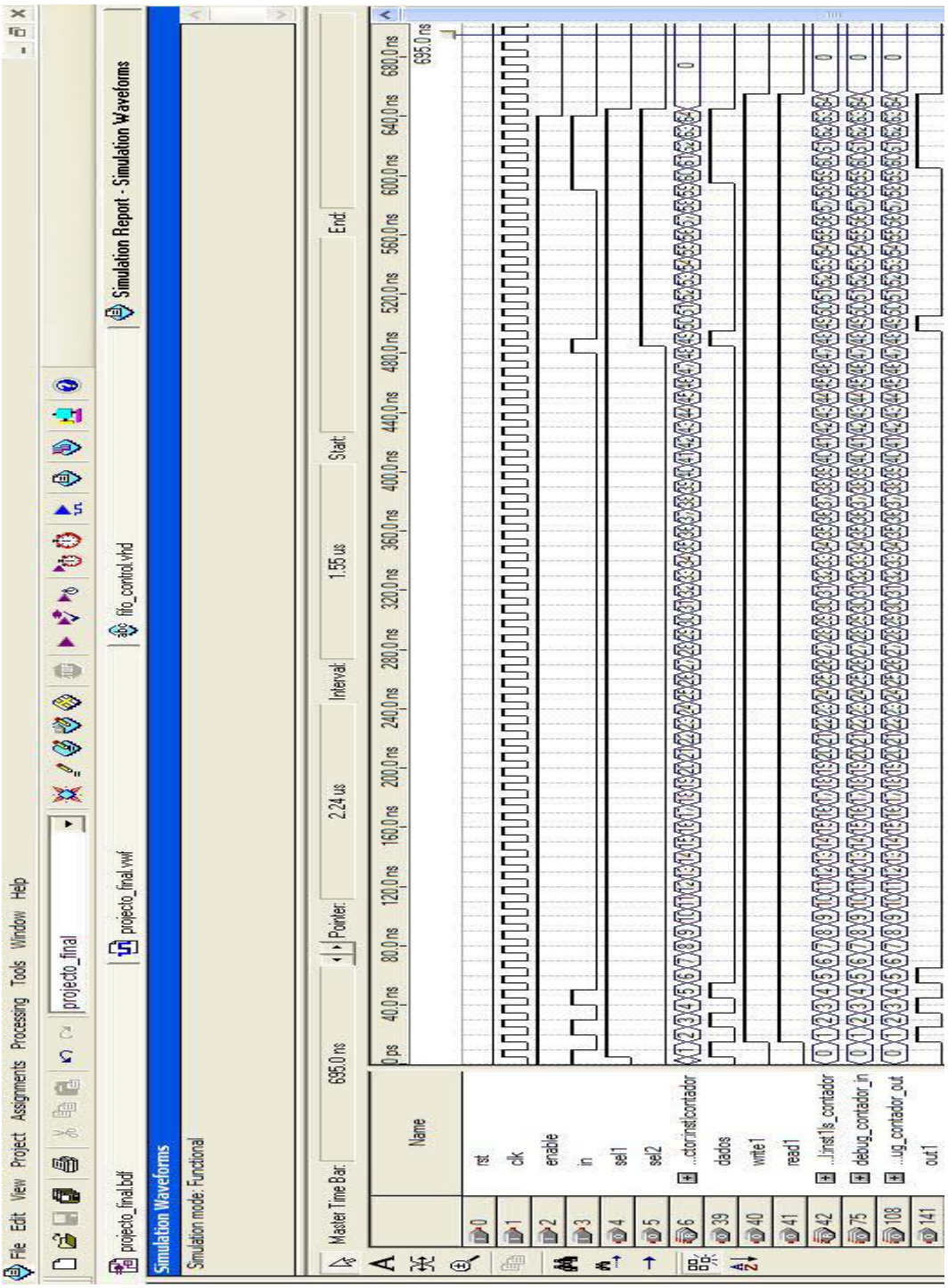

Figura 45: Resultados funcionais do bloco fifo_control no intervalo [0;0,69] µs

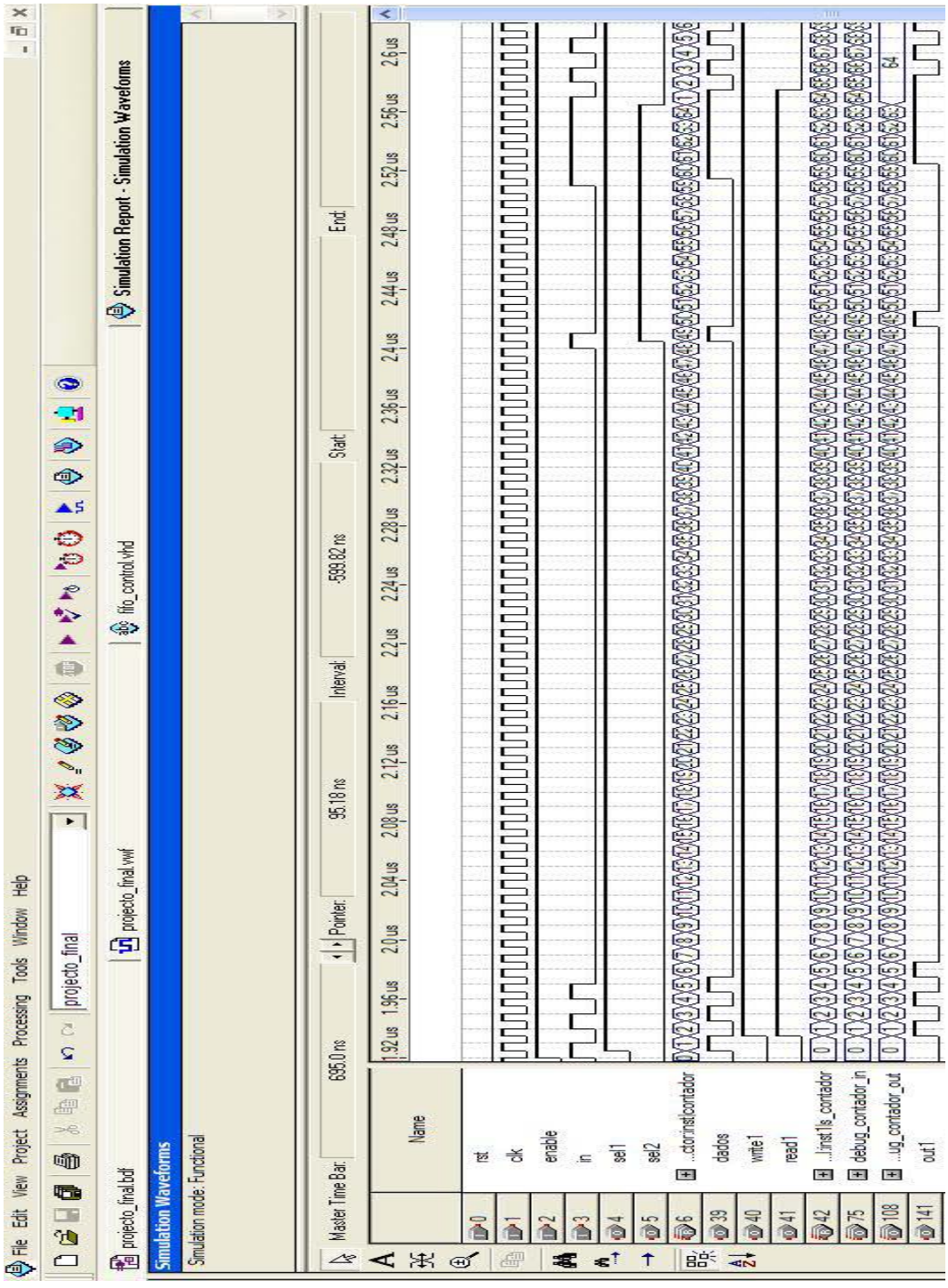

Figura 46: Resultados funcionais do bloco fifo_control no intervalo [1,92;2,62] µs

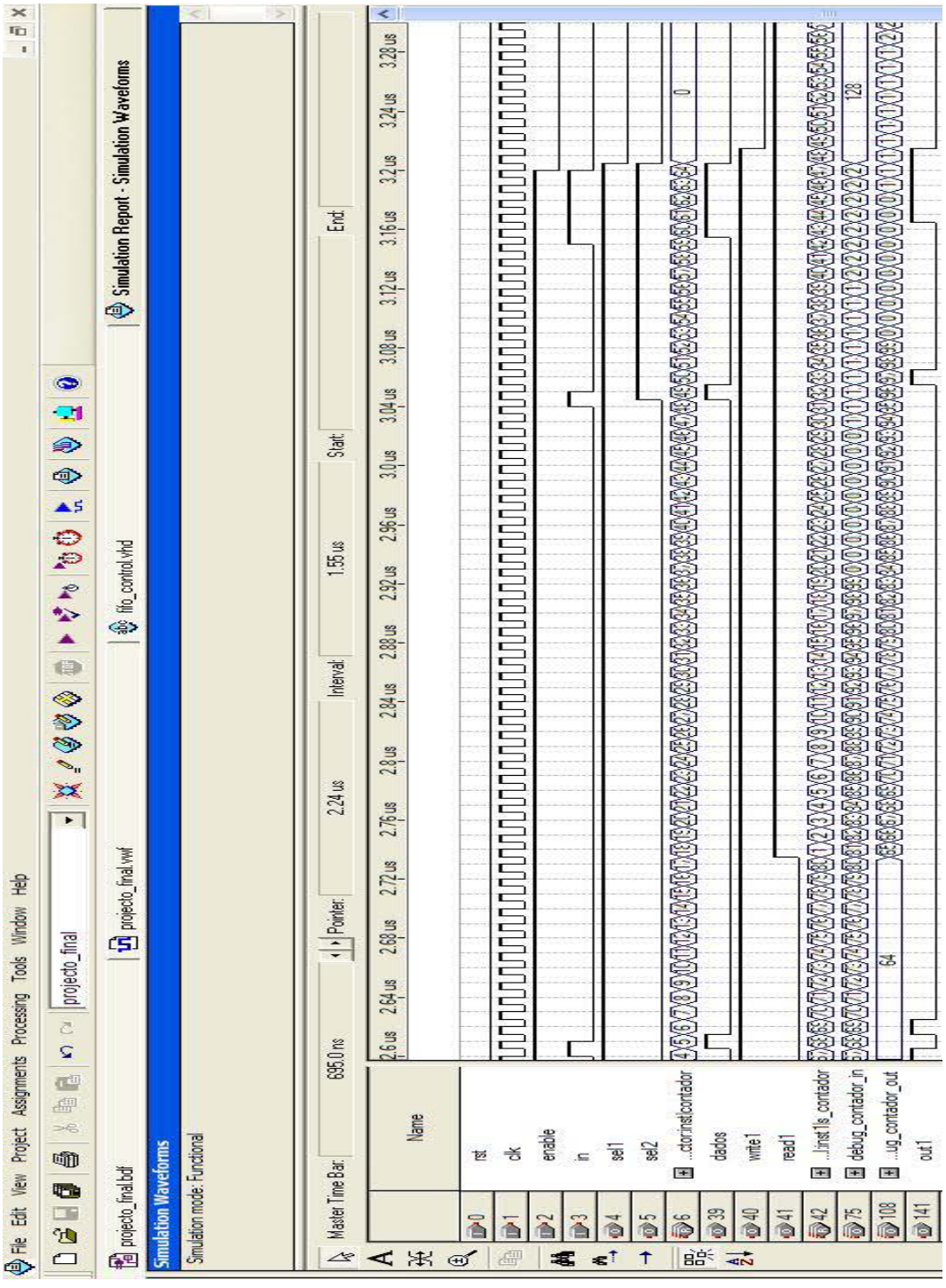

Figura 47: Resultados funcionais do bloco fifo_control no intervalo [2,6;3,3] µs

4.4 – Fifo_control2

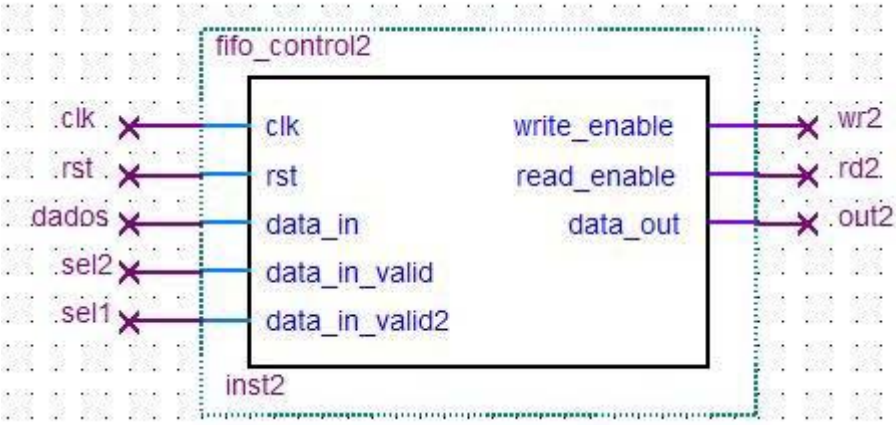

Figura 48: fifo_control2 implementado no Quartus II

4.4.1 – Características do fifo_control2

Tabela 9: Características do bloco fifo_control2

Portas de entrada	clk
	rst
	data_in
	data_in_valid
	data_in_valid2
Portas de saída	write_enable
	read_enable
	data_out
Sinais	s_cont
	s_contador_in
	s_contador_out
	flag
	s_read_enable
	s_contador_read_enable
	s_cont_data_in_valid2

4.4.2 – Descrição das operações do fifo_control2

- Neste bloco estão definidos quatro processos, todos decorrendo simultaneamente.
- **1º processo:** sempre que reset esteja activo (‘1’), o sinal s_contador_read_enable é remetido a ‘0’, senão por cada transição ascendente do sinal s_read_enable o sinal s_contador_read_enable é incrementado.
- Se o sinal flag for ‘0’, então, o sinal s_contador_read_enable é remetido a ‘0’.
- **2º processo:** sempre que reset esteja activo (‘1’), os sinais s_read_enable, s_contador_in, s_contador_out, s_cont e as saídas write_enable e read_enable são remetidos a ‘0’.
- Por cada transição ascendente do clock são efectuadas as seguintes operações:
- Se a entrada data_in_valid for ‘1’, então:
- A saída write_enable é remetida a ‘1’, a saída data_out recebe o valor de data_in e o sinal s_contador_in é incrementado.
- Se o valor do sinal flag for ‘1’, então: o sinal s_cont é incrementado.
- Se o valor do sinal s_cont estiver compreendido entre 64 e 80, a saída read_enable toma o valor ‘1’, o sinal s_read_enable recebe o valor ’0’ e o sinal s_contador_out é incrementado.
- Se o sinal s_cont atingir o valor 80, a saída read_enable toma o valor ‘0’, os sinais s_read_enable e s_cont tomam o valor ‘1’.
- Se o valor do sinal flag for ‘0’, o sinal s_cont toma o valor ‘0’.
- Se o valor de data_in_valid for ‘0’, então, as saídas data_out e write_enable tomam o valor ‘0’.
- Se o valor do sinal flag for ‘1’, então: o sinal s_cont é incrementado.
- Se o valor do sinal s_cont estiver compreendido entre 64 e 80, a saída read_enable toma o valor ‘1’, o sinal s_read_enable recebe o valor ’0’ e o sinal s_contador_out é incrementado.
- Se o sinal s_cont atingir o valor 80, a saída read_enable toma o valor ‘0’, os sinais s_read_enable e s_cont tomam o valor ‘1’.
- Se o valor do sinal flag for ‘0’, o sinal s_cont toma o valor ‘0’.
- Se os valores dos sinais s_contador_in e s_contador_out forem iguais então, os sinais s_contador_in e s_contador_out tomam o valor ‘0’.
- **3º processo:** sempre que reset esteja activo (‘1’), o sinal flag é remetido a ‘0’, senão por cada transição ascendente do sinal data_in_valid2, o sinal flag toma o valor ‘1’.
- Se o valor do sinal s_cont_data_in_valid2 for igual ao valor de 64 vezes o valor do sinal s_contador_read_enable, e, o valor do sinal

s_cont_data_in_valid2 for maior que zero, então o sinal flag toma o valor ‘0’.

- **4º processo:** sempre que reset esteja activo (‘1’), o sinal s_cont_data_in_valid2 é remetido a ‘0’, senão por cada transição ascendente do clock, são efectuadas as seguintes operações:
- Se o valor da entrada data_in_valid2 for ‘1’, o valor do sinal s_cont_data_in_valid2 é incrementado.
- Se o valor do sinal flag for ‘0’, o sinal s_cont_data_in_valid2 toma o valor ‘0’.

Nas figuras 49 e 50, está de novo a chegada de um bloco de 64 bits na entrada in e são visíveis os sinais write2 e read2 inactivos ‘0’ inicialmente, o contador vai sendo desde o início incrementado, enquanto o contador contador_in2 e contador_out2 estão inactivos. Quando o sinal sel2 fica activo ‘1’ então, o sinal de write2 também fica activo ‘1’ e fica inactivo ‘0’ quando sel2 fica inactivo. Quando o sinal write2 está activo o contador contador_in2 é incrementado pois estão a ser escritos dados. O sinal read2 é independente do sinal de write2, pois o ritmo de leitura tem que ser muito específico, isto é, este ritmo está adequado ao ritmo de leitura do bloco anterior (fifo_control). Sendo assim, criou-se um processo onde existe uma alteração do valor de uma flag, ou seja, por cada transição ascendente do sinal data_in_valid2 (sel1), a flag toma o valor ‘1’.

Foi criado um novo processo para criar um contador (s_cont_data_data_in_valid2) que é incrementado quando a entrada data_in_valid2 é ‘1’. Num novo processo é contabilizado no contador s_contador_read_enable o número de vezes que o sinal s_read_enable possui uma transição ascendente. Se o contador s_cont_data_data_in_valid2 possuir um valor que seja igual a 64 vezes o valor do contador s_contador_read_enable e s_cont_data_data_in_valid2 for maior que zero, então a flag toma o valor zero no processo onde é alterado o valor da flag. Isto garante o valor zero da flag quando todos os bits relativos ao prefixo já tiverem sido escritos e lidos. Quando read_enable está activo ‘1’ o contador contador_out2 será incrementado e quando os valores do contador contador_in2 e contador_out2 sejam iguais, então são inicializados a zero pois todos os bits já foram lidos. Este processo onde são criados contadores e o sinal da flag foi apenas um método de criar a saída de read_enable de acordo com a saída read_enable do bloco fifo_control pois estas saídas têm que estar sincronizadas, ou seja, quando uma está activa a outra está inactiva e vice-versa. No anexo C encontra-se o código VHDL do bloco fifo_control2.

4.4.3 – Resultados da simulação funcional do fifo_control2

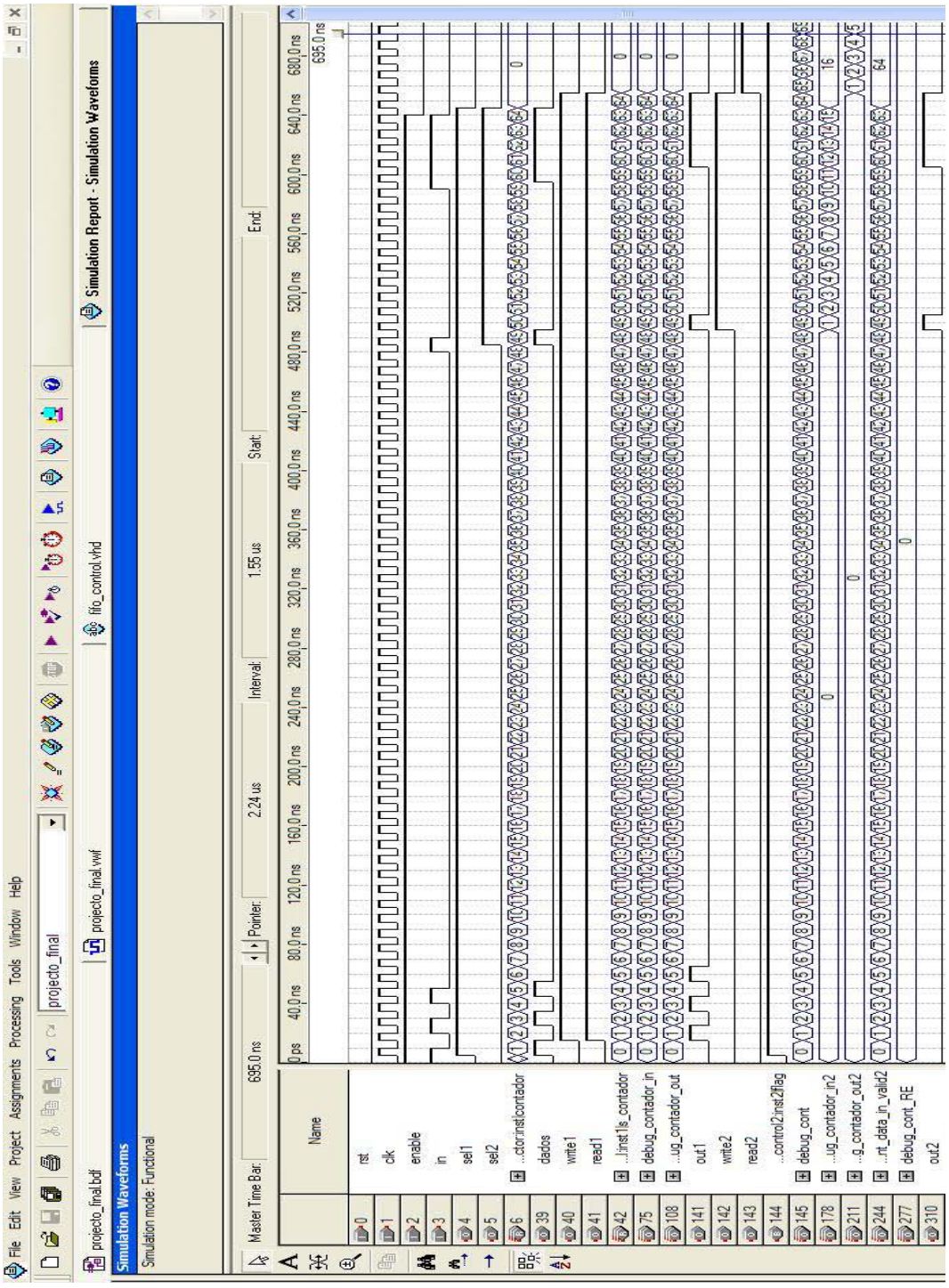

Figura 49: Resultados funcionais do bloco fifo_control2 no intervalo [0;0,695] µs

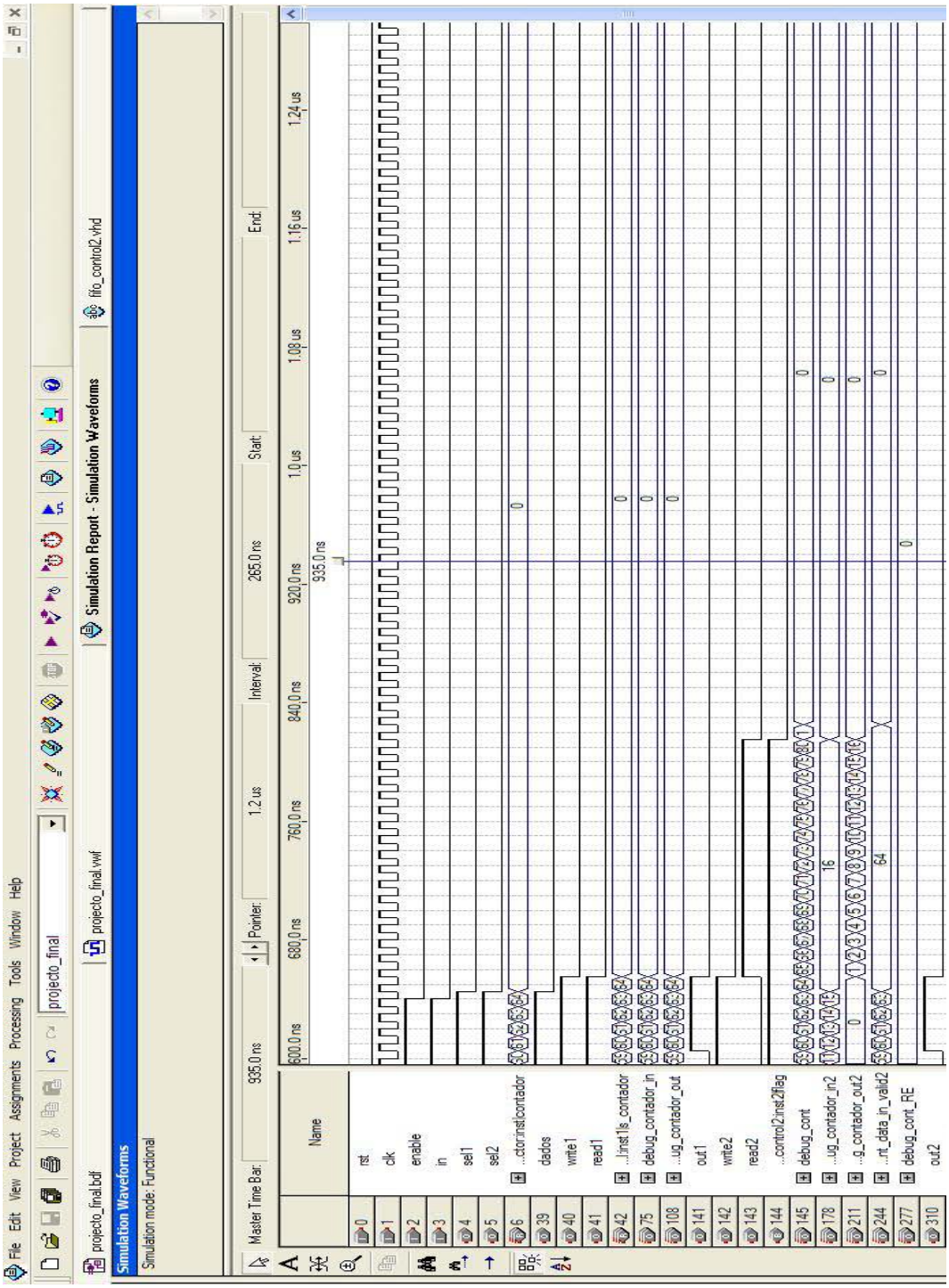

Figura 50: Resultados funcionais do bloco fifo_control2 no intervalo [0,6;1,28] µs

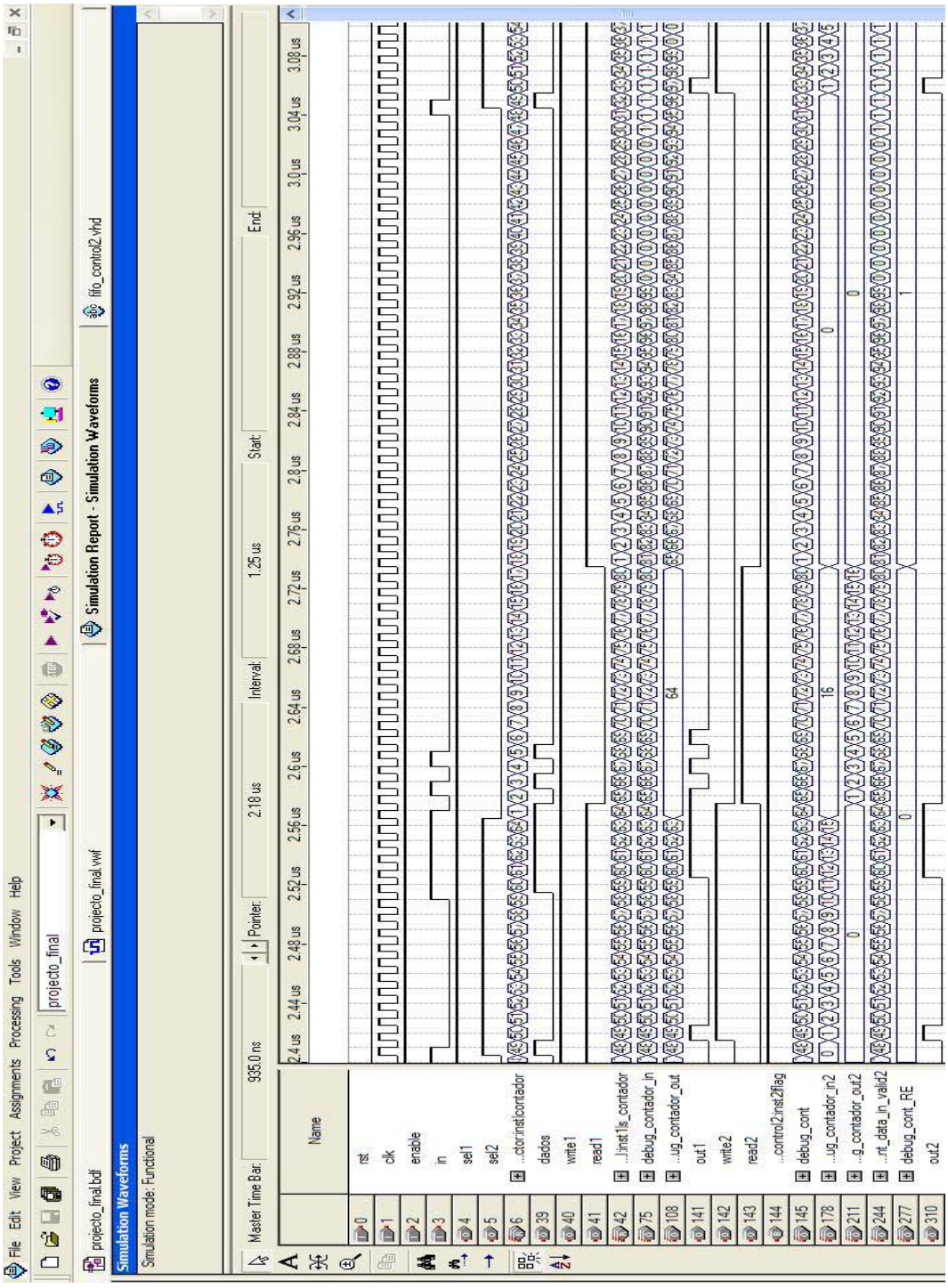

Figura 51: Resultados funcionais do bloco fifo_control2 no intervalo [2,4;3,1] µs

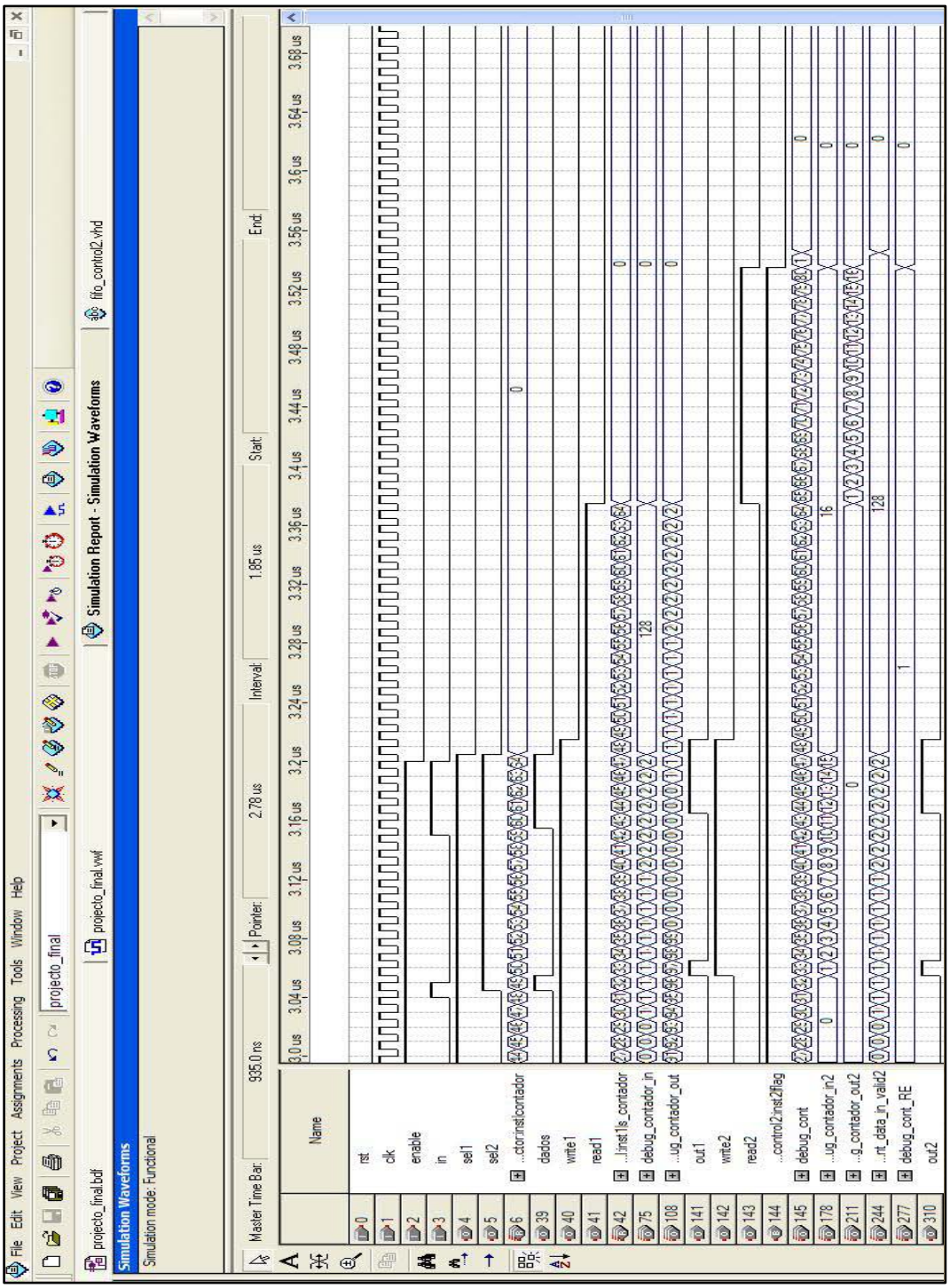

Figura 52: Resultados funcionais do bloco fifo_control2 no intervalo [3;3,7] µs

4.5 – Fifo

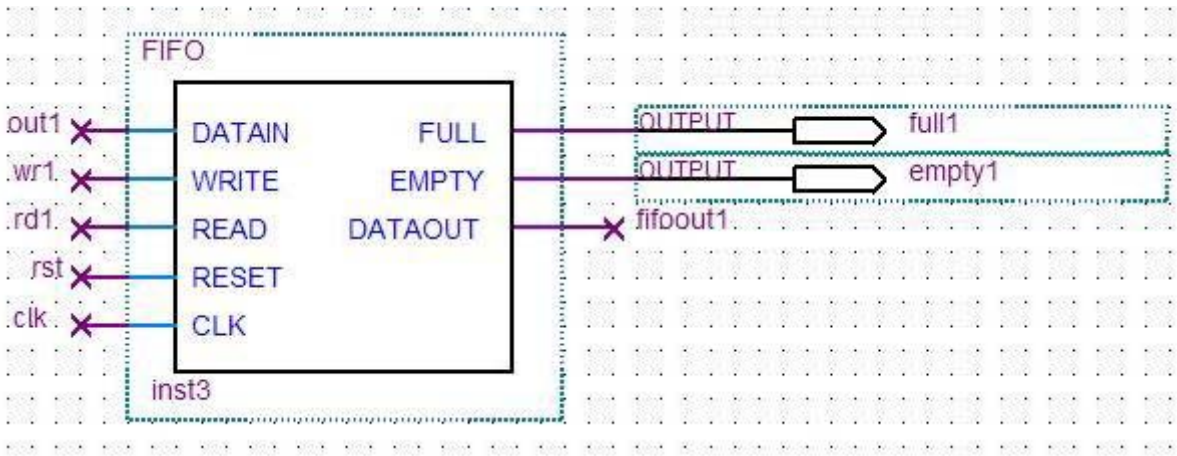

Figura 53: fifo implementado no Quartus II

4.5.1 – Características do fifo

Tabela 10: Características do bloco fifo

Portas de entrada	DATAIN
	WRITE
	READ
	RESET
	CLK
Portas de saída	FULL
	EMPTY
	DATAOUT
Variáveis	FIFO
	BOTTOM
	TOP
	C_SIZE
	FLAG_EMPTY
	FLAG_FULL
Array	FIFO_TYPE

4.5.2 – Descrição das operações do fifo

- Por cada transição ascendente de relógio são efectuadas as seguintes operações.
- Sempre que RESET esteja activo (‘1’), as portas de saída DATAOUT e FULL, as variáveis FLAG_FULL, TOP, BOTTOM e C_SIZE são remetidas a zero e a variável FLAG_EMPTY e a saída EMPTY tomam o valor ‘1’.

- Se a entrada WRITE for ‘1’, a variável C_SIZE for menor ou igual a 16k e a variável FLAG_FULL for diferente de ‘1’, então, a variável FIFO na posição indicada por BOTTOM recebe o valor da entrada DATAIN.
- A variável BOTTOM é incrementada.
- Se a variável BOTTOM for maior que 16k então é remetida ao valor ‘0’.
- Se a variável C_SIZE for menor que 16k é incrementada, senão, a saída FULL e a variável FLAG_FULL tomam o valor ‘1’.
- A saída EMPTY e variável FLAG_EMPTY recebem o valor ‘0’.
- Se a entrada READ for ‘1’, a variável C_SIZE for maior que zero e a variável FLAG_EMPTY for diferente de ‘1’, então, a saída DATAOUT recebe o valor da variável FIFO na posição indicada por TOP, senão a saída DATAOUT recebe o valor ‘0’.
- A variável TOP é incrementada.
- Se a variável TOP for maior que 16k então é remetida ao valor ‘0’.
- A variável C_SIZE é decrementada.
- Se a variável C_SIZE for igual a zero, então, a saída EMPTY e a variável FLAG_EMPTY tomam o valor ‘1’, e, as variáveis TOP E BOTTOM tomam o valor ‘0’.
- A saída FULL e variável FLAG_FULL recebem o valor ‘0’.

Na figura 54 está exemplificado o exemplo do fifo receber um bloco de 64 bits. Como estão activos ‘1’ os sinais de write e read o fifo limita-se a receber e enviar a informação, sendo que o estado encontra-se vazio porque não está a ocorrer nenhum armazenamento. No final quando toda a informação já tiver sido lida os sinais de write e read tomam o valor zero.

Nas figuras 55 e 56 dois blocos de 64 bits estão na entrada in sendo que o primeiro é lido de imediato, ou seja, os sinais write e read estão activos. O segundo bloco é escrito no fifo mas só é lido passado 16 bits, ou seja, neste período o fifo vai armazenar informação, sendo que o estado deixa de ser vazio, estando a saída empty a ‘0’.

O fifo terá uma capacidade de armazenamento de 16k bits, o que será suficiente para uma trama completa de 64k bits. Como existe um espaçamento temporal entre transmissões ($\approx 10^{-2}$, 10^{-1} s), e esse tempo será maior do que o tempo que demorará este fifo a esvaziar ($\approx 10^{-4}$ s), esta capacidade de 16k bits será a ideal para este caso. No anexo D encontra-se o código VHDL do bloco fifo.

4.5.3 – Resultados da simulação funcional do fifo

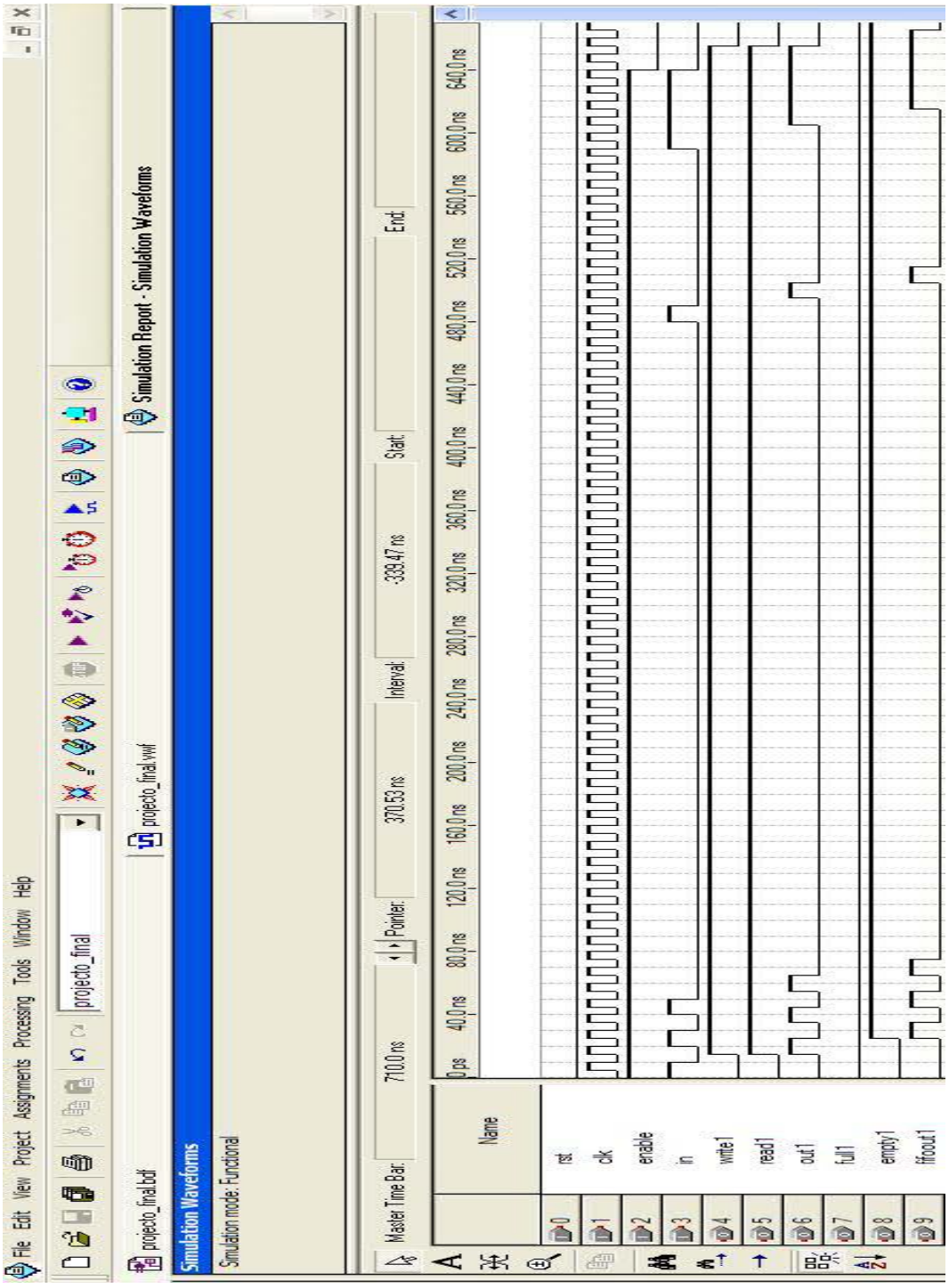

Figura 54: Resultados funcionais do bloco fifo no intervalo [0;0,65] µs

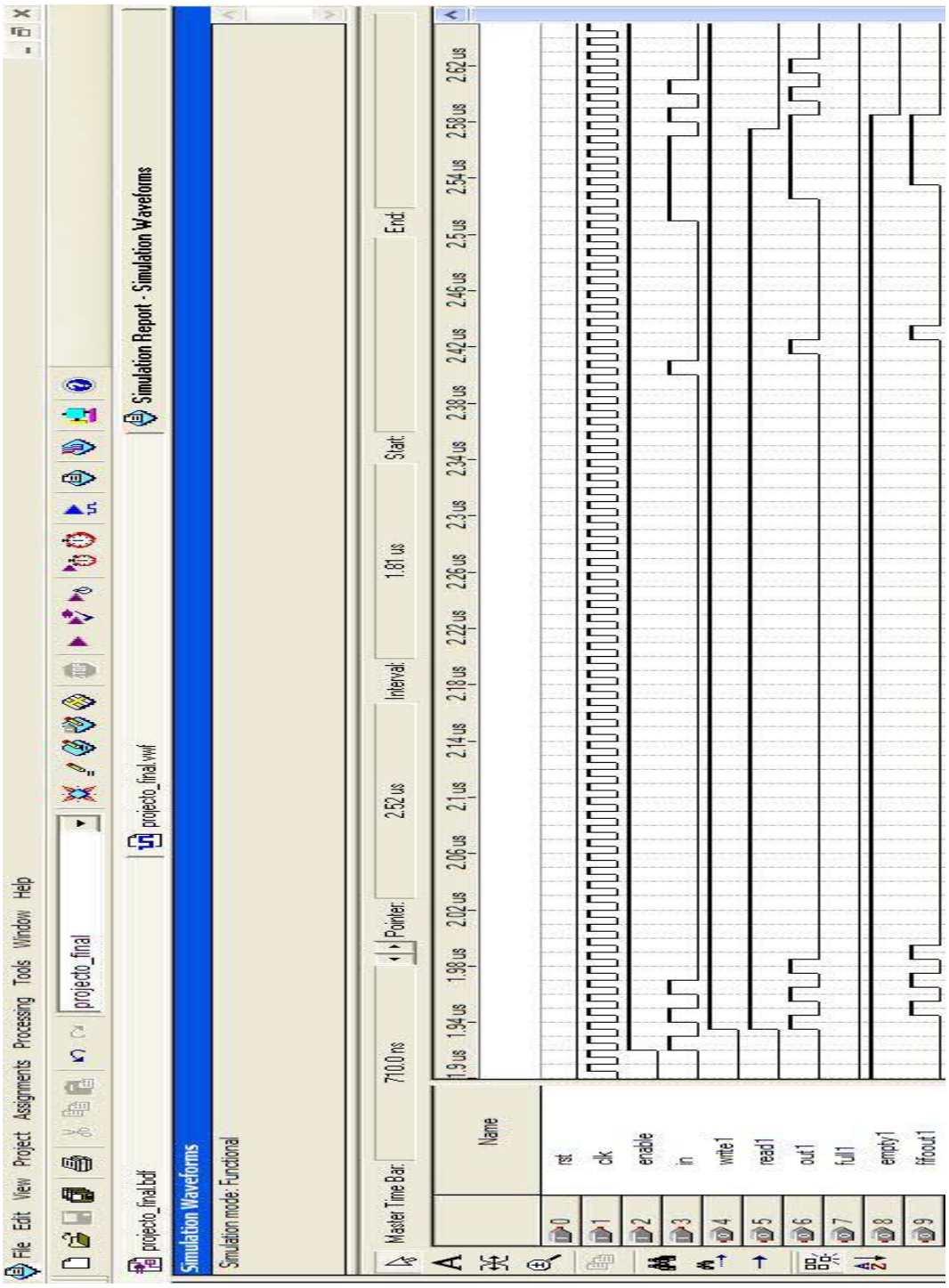

Figura 55: Resultados funcionais do bloco fifo no intervalo [1,9;2,64] µs

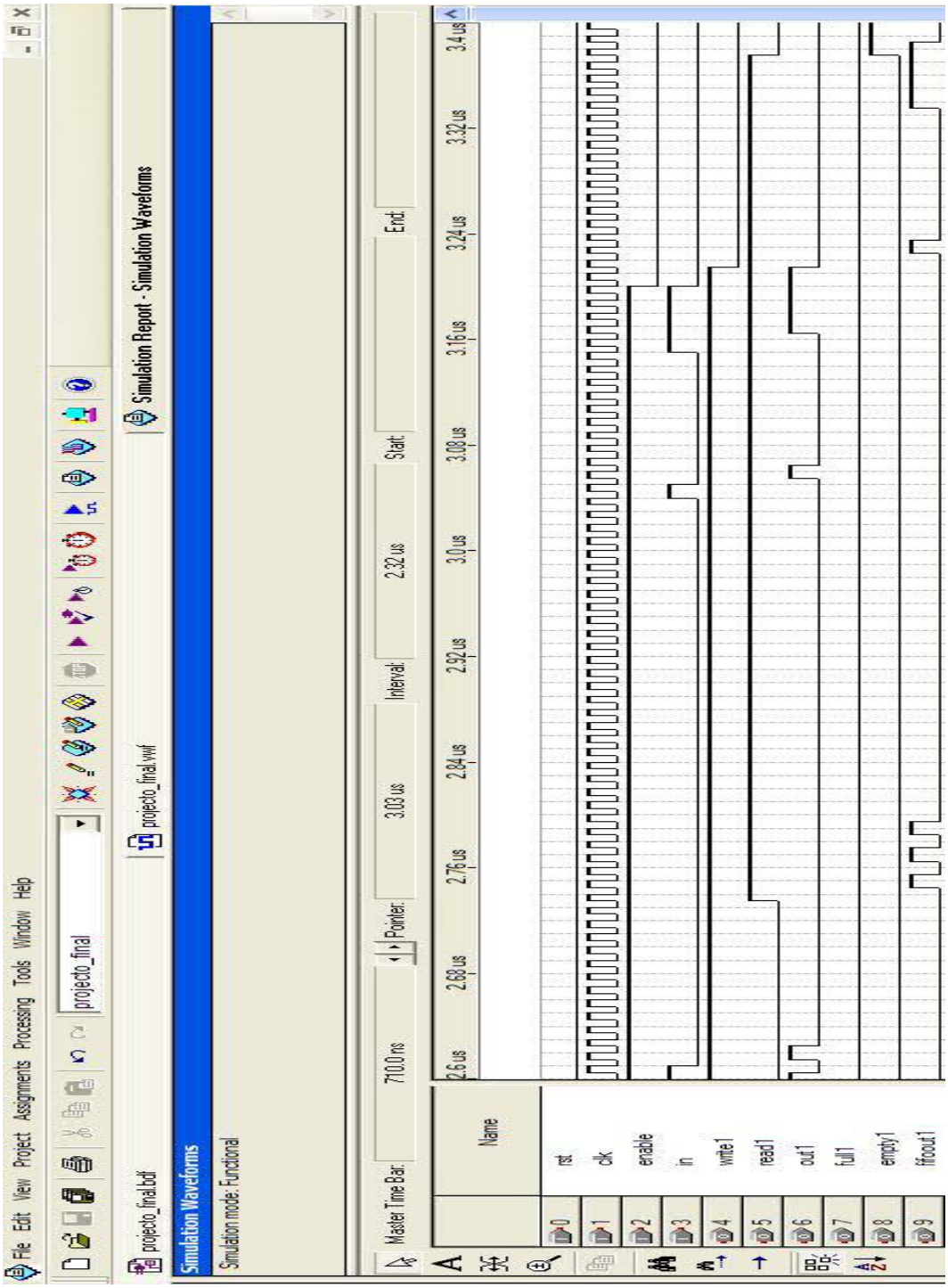

Figura 56: Resultados funcionais do bloco fifo no intervalo [2,6;3,4] µs

4.6 – Fifo2

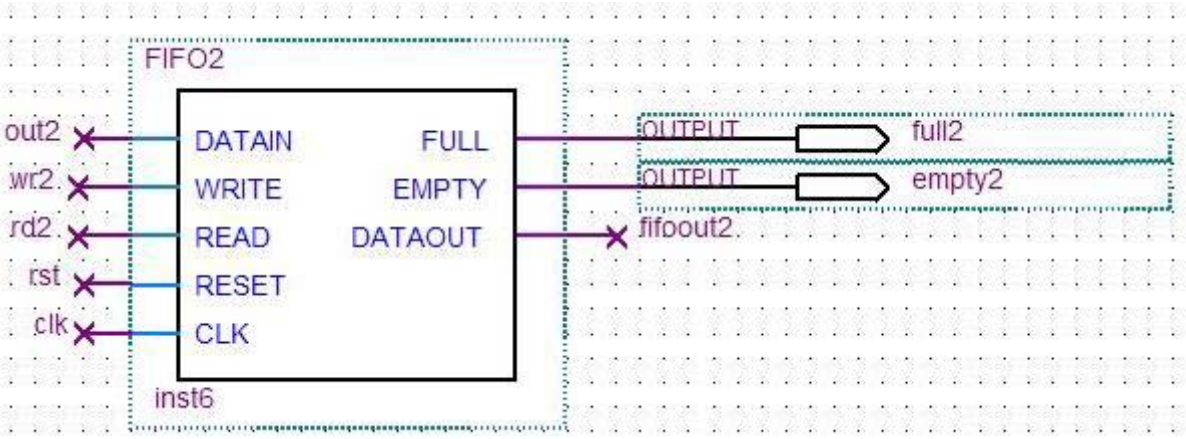

Figura 57: fifo2 implementado no Quartus II

4.6.1 – Características do fifo2

Tabela 11: Características do bloco fifo2

Portas de entrada	DATAIN
	WRITE
	READ
	RESET
	CLK
Portas de saída	FULL
	EMPTY
	DATAOUT
Variáveis	FIFO
	BOTTOM
	TOP
	C_SIZE
	FLAG_EMPTY
	FLAG_FULL
Array	FIFO_TYPE

4.6.2 – Descrição das operações do fifo2

- Por cada transição ascendente de relógio são efectuadas as seguintes operações.

- Sempre que RESET esteja activo (‘1’), as portas de saída DATAOUT e FULL, as variáveis FLAG_FULL, TOP, BOTTOM e C_SIZE são remetidas a zero e a variável FLAG_EMPTY e a saída EMPTY tomam o valor ‘1’.
- Se a entrada WRITE for ‘1’, a variável C_SIZE for menor ou igual a 3,3k e a variável FLAG_FULL for diferente de ‘1’, então, a variável FIFO na posição indicada por BOTTOM recebe o valor da entrada DATAIN.
- A variável BOTTOM é incrementada.
- Se a variável BOTTOM for maior que 3,3k então é remetida ao valor ‘0’.
- Se a variável C_SIZE for menor que 3,3k é incrementada, senão, a saída FULL e a variável FLAG_FULL tomam o valor ‘1’.
- A saída EMPTY e variável FLAG_EMPTY recebem o valor ‘0’.
- Se a entrada READ for ‘1’, a variável C_SIZE for maior que zero e a variável FLAG_EMPTY for diferente de ‘1’, então, a saída DATAOUT recebe o valor da variável FIFO na posição indicada por TOP, senão a saída DATAOUT recebe o valor ‘0’.
- A variável TOP é incrementada.
- Se a variável TOP for maior que 3,3k então é remetida ao valor ‘0’.
- A variável C_SIZE é decrementada.
- Se a variável C_SIZE for igual a zero, então, a saída EMPTY e a variável FLAG_EMPTY tomam o valor ‘1’, e, as variáveis TOP E BOTTOM tomam o valor ‘0’.
- A saída FULL e variável FLAG_FULL recebem o valor ‘0’.

Na figura 58 está exemplificado o exemplo do fifo2 receber um bloco de 64 bits. É de verificar que o sinal write2 apenas fica activo quando se trata da zona de dados do prefixo, ou seja, dos últimos 16 bits, sendo que quando este sinal fica inactivo ‘0’, o sinal de read2 estará activo durante o mesmo período de tempo, libertando os dados para a saída fifoout2 ficando o fifo2 vazio como se pode verificar pelo sinal empty2 ao ser activado ‘1’.

Nas figuras 59 e 60, dois blocos de 64 bits estão na entrada in sendo que o processo anteriormente descrito se repete, havendo agora a diferença de ritmo entre write2 e read2 imposto pelo bloco fifo_control2.

O fifo2 terá uma capacidade de armazenamento de 3,3k bits, o que será suficiente para uma trama completa de 64k bits. Como existe um desfasamento entre escrita e leitura, e sabendo que este fifo armazenará blocos de 16 bits, então, a cada 5 blocos de 16 bits haverá um atraso de 1 bloco de 16 bits, isto faz com que para uma trama de 64k bits, a capacidade de 3,3k seja suficiente para armazenar todos os dados pretendidos. No anexo E encontra-se o código VHDL do bloco fifo2.

4.6.3 – Resultados da simulação funcional do fifo2

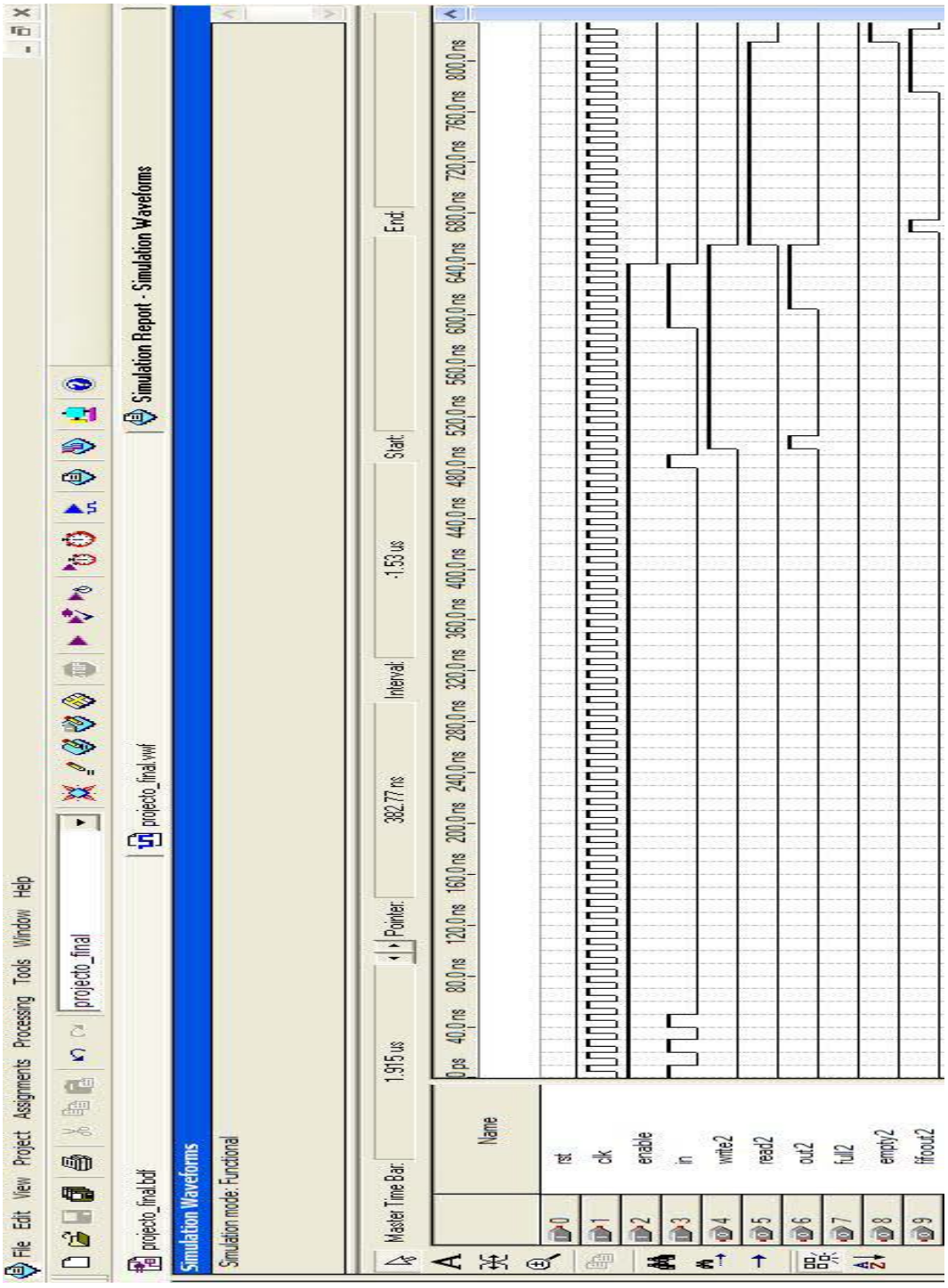

Figura 58: Resultados funcionais do bloco fifo2 no intervalo [0;0,82] µs

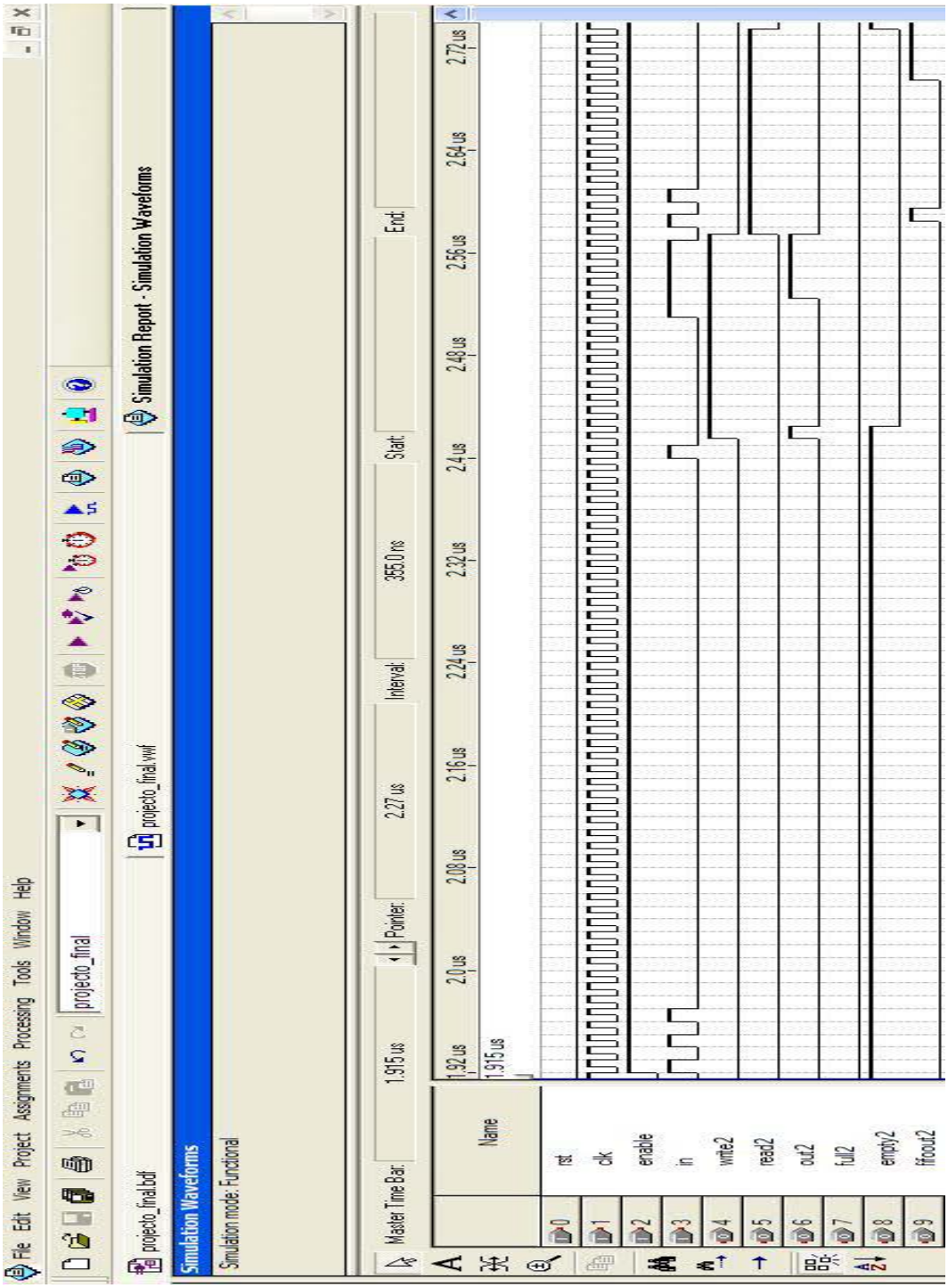

Figura 59: Resultados funcionais do bloco fifo2 no intervalo [1,92;2,74] µs

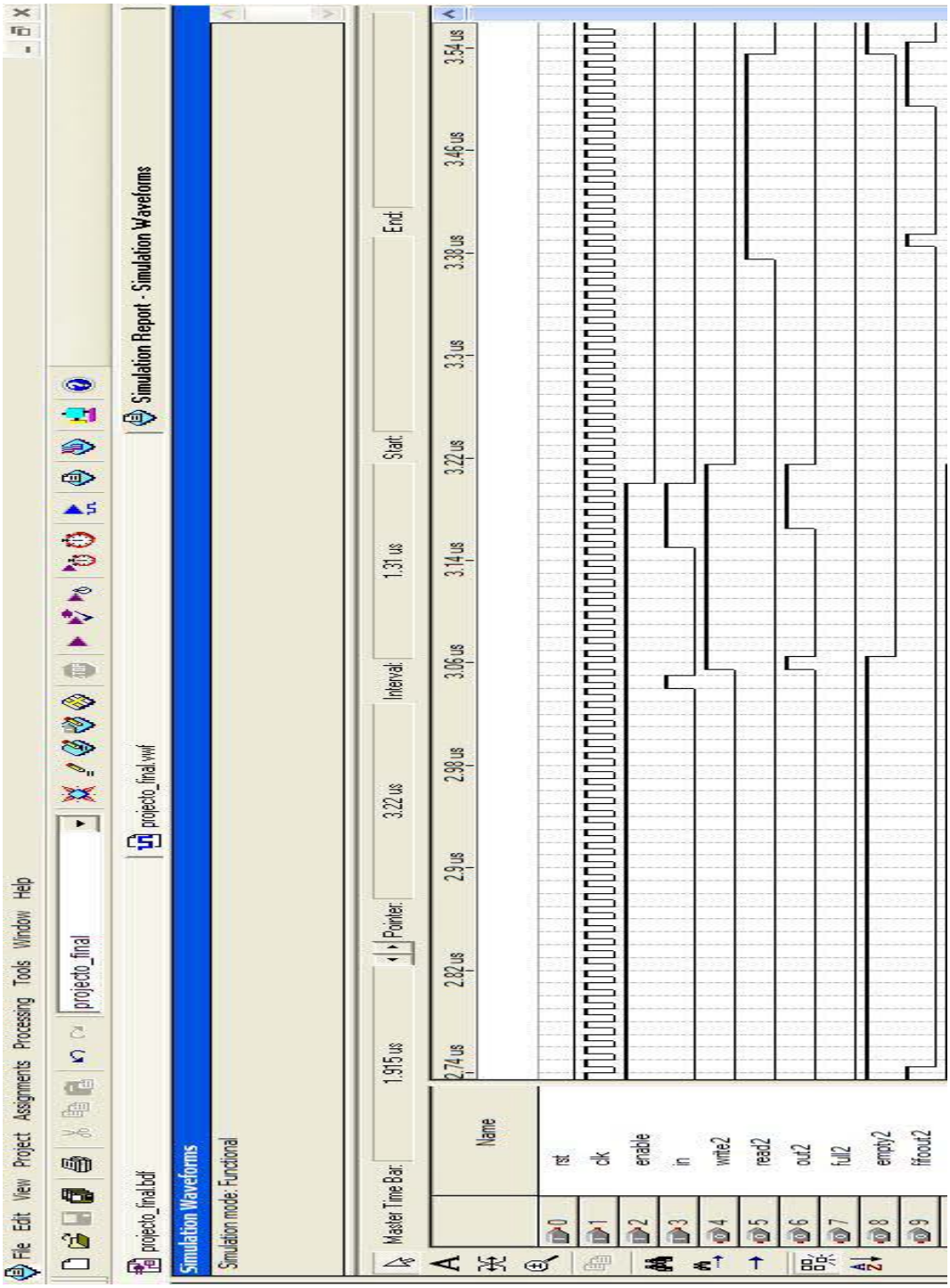

Figura 60: Resultados funcionais do bloco fifo2 no intervalo [2,74;3,56] µs

4.7 − LPM_SHIFTREG

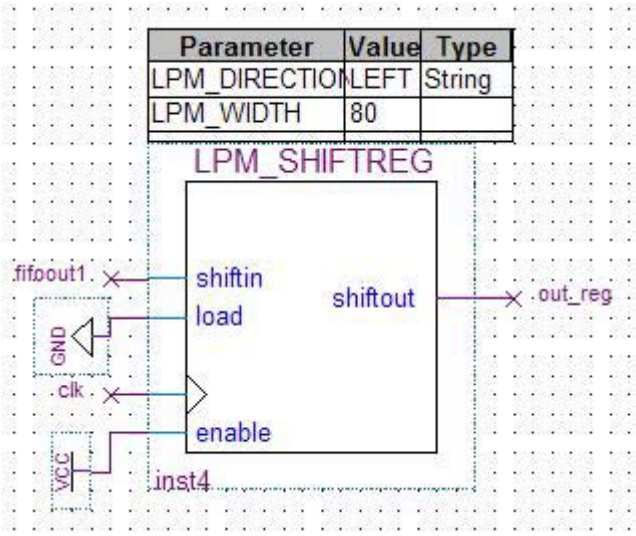

Figura 61: lpm_shiftreg implementado no Quartus II

4.7.1 − Características do lpm_shiftreg

Tabela 12: Características do bloco lpm_shiftreg

Portas de entrada	shiftin
	load
	clk
	enable
Portas de saída	shiftout
Parâmetros	LPM_DIRECTION = LEFT
	LPM_WIDTH = 80

4.7.2 − Descrição das operações do lpm_shiftreg

Este bloco provém da megafunção lpm_shiftreg já existente no Quartus II sendo que é parametrizável de acordo com o objectivo pretendido.

- É verificado o estado da entrada enable e se for activa ‘1’, então, o bloco procederá às operações nele definidas.
- Por cada transição ascendente de relógio são efectuadas as seguintes operações.
- Os dados da entrada shiftin vão entrando no bloco e vão avançando, percorrendo o registo de tamanho 80, chegando à saída shiftout.
- O objectivo é introduzir um atraso no sinal de tamanho igual ao do registo.

Como se pode verificar nas figuras 62, 63 e 64 o bloco lpm_shiftreg apenas tem a função de introduzir um atraso na saída fifoout1. Este atraso é aplicado, apenas para haver sincronismo entre as saídas fifoout1 e fifoout2. Este atraso será de 80 bits definido no parâmetro lpm_width e poderá ser verificado na saída out_reg onde será uma cópia dos dados de fifoout1 apenas atrasados 80 ciclos relógio ou 80 bits. No anexo F encontra-se o código VHDL do bloco lpm_shiftreg (código da Altera parametrizável).

4.7.3 − Resultados da simulação funcional do lpm_shiftreg

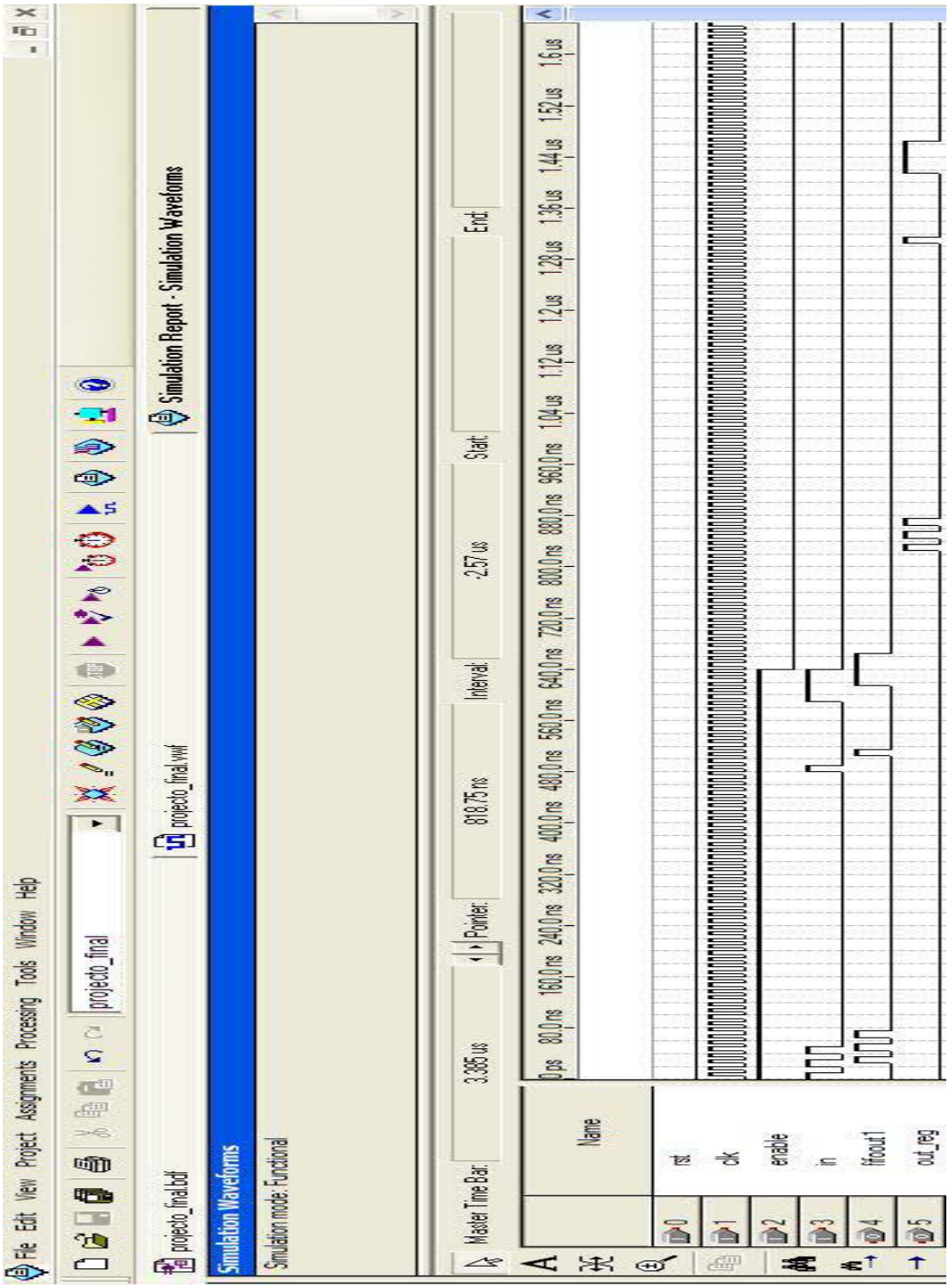

Figura 62: Resultados funcionais do bloco lpm_shiftreg no intervalo [0;1,6] µs

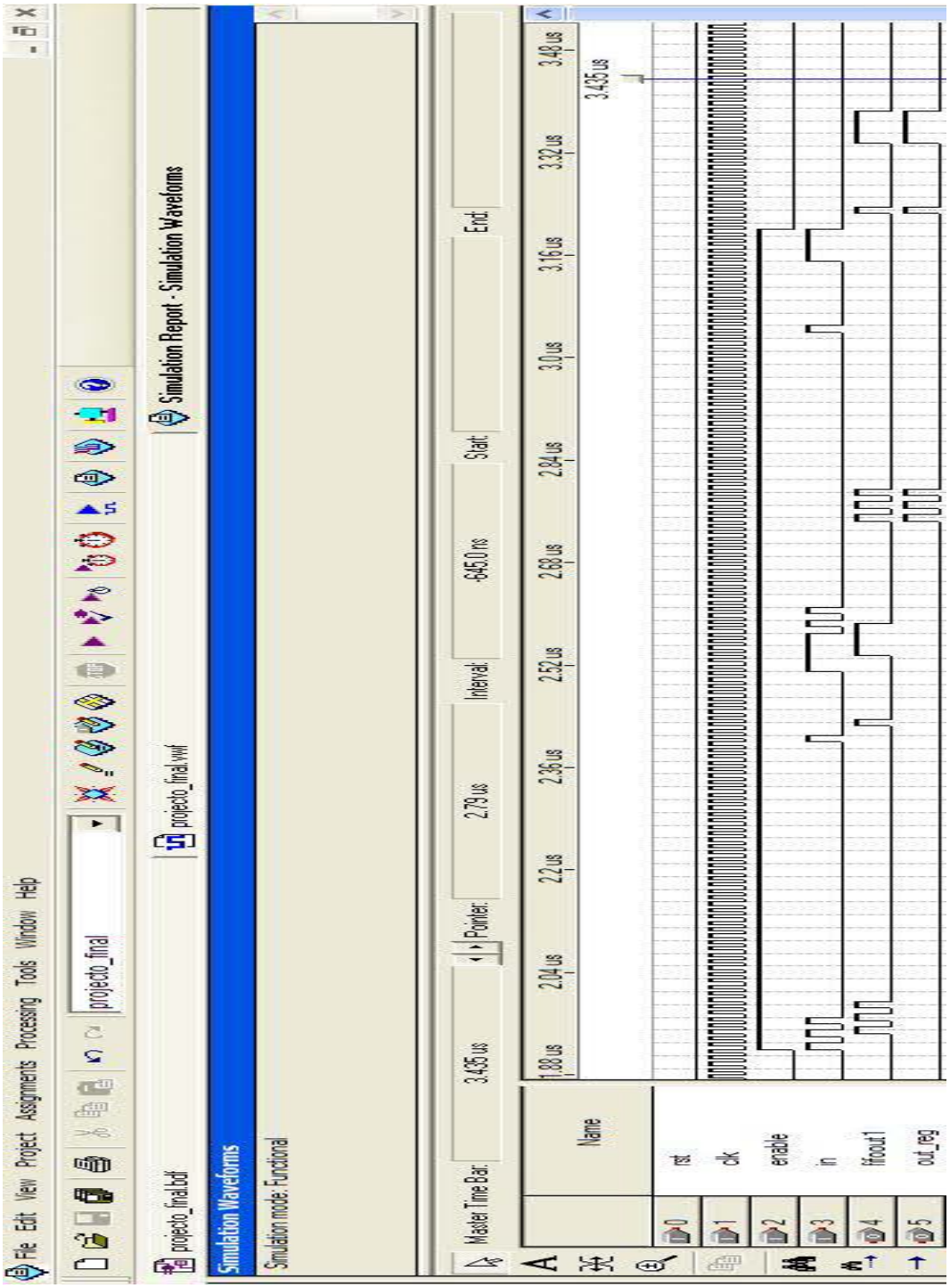

Figura 63: Resultados funcionais do bloco lpm_shiftreg no intervalo [1,88;3,48] µs

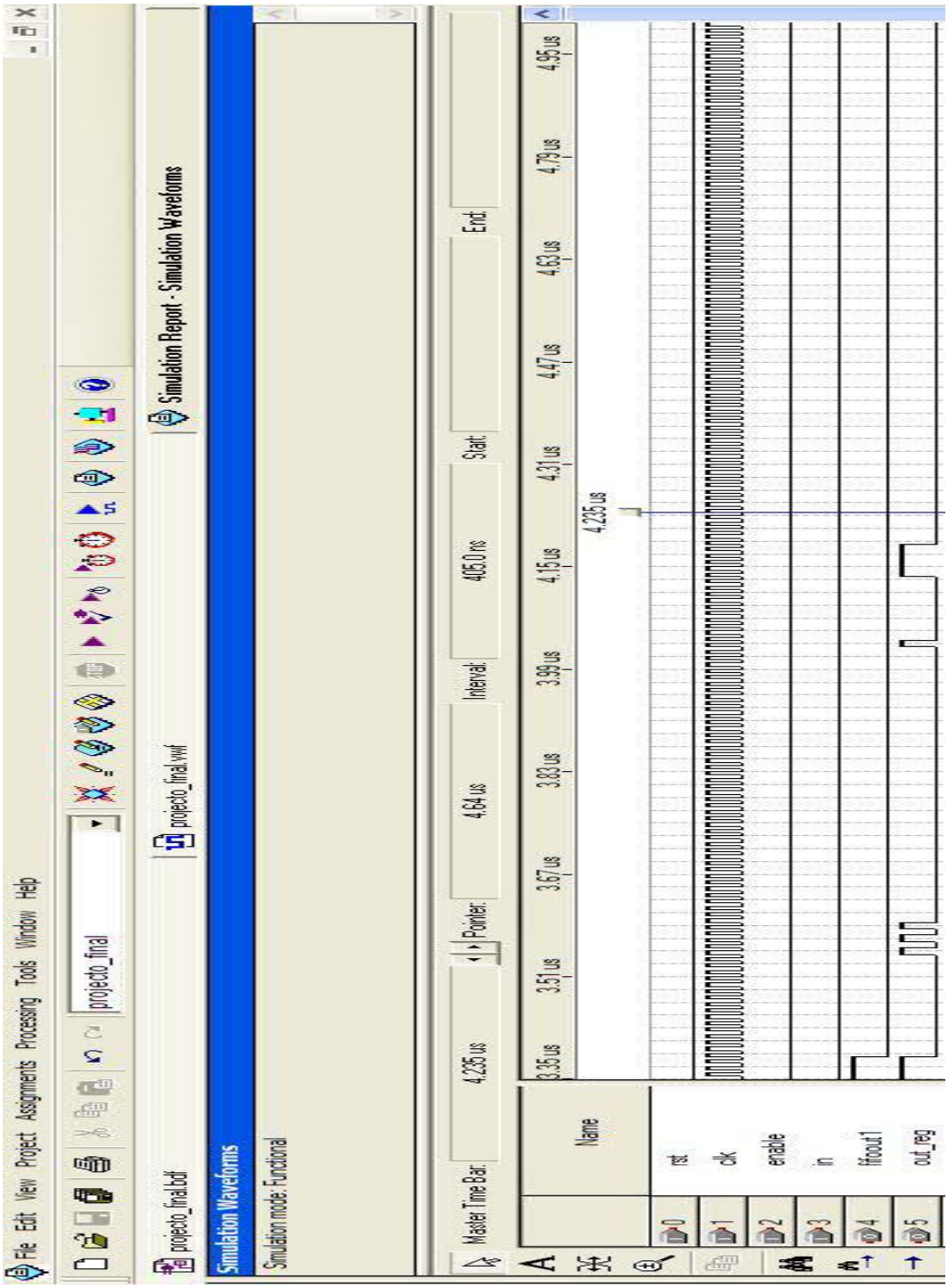

Figura 64: Resultados funcionais do bloco lpm_shiftreg no intervalo [3,35;4,95] µs

4.8 − Or

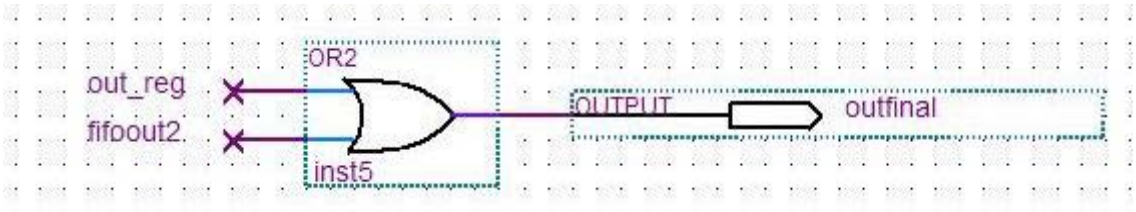

Figura 65: Or implementado no Quartus II

4.8.1 − Características do or

Tabela 13: Características do bloco or

Portas de entrada	out_reg
	fifoout2
Portas de saída	outfinal

4.8.2 − Descrição das operações do or

Este bloco provém da primitiva lógica or2 já existente no Quartus II.

- O bloco recebe dois sinais, neste caso out_reg e fifoout2, cada um numa porta de entrada, e produz no sinal de saída outfinal, a soma das entradas referidas anteriormente.

Na figura 66 pode-se verificar a chegada dum bloco de 64 bits à entrada in, e posteriormente depois da passagem por todos os blocos anteriormente descritos obtêm-se dois sinais de nome out_reg e fifoout2 que são somados obtendo-se finalmente a saída, outfinal, em que o processo de adição de prefixo aos blocos está implementado.

Nas figuras 67 e 68, pode-se ver o exemplo para 2 blocos de 64 bits cada, em que à saída surgem 160 bits correspondendo a 80 bits de cada novo bloco já com o prefixo incluído.

4.8.3 − Resultados da simulação funcional do or

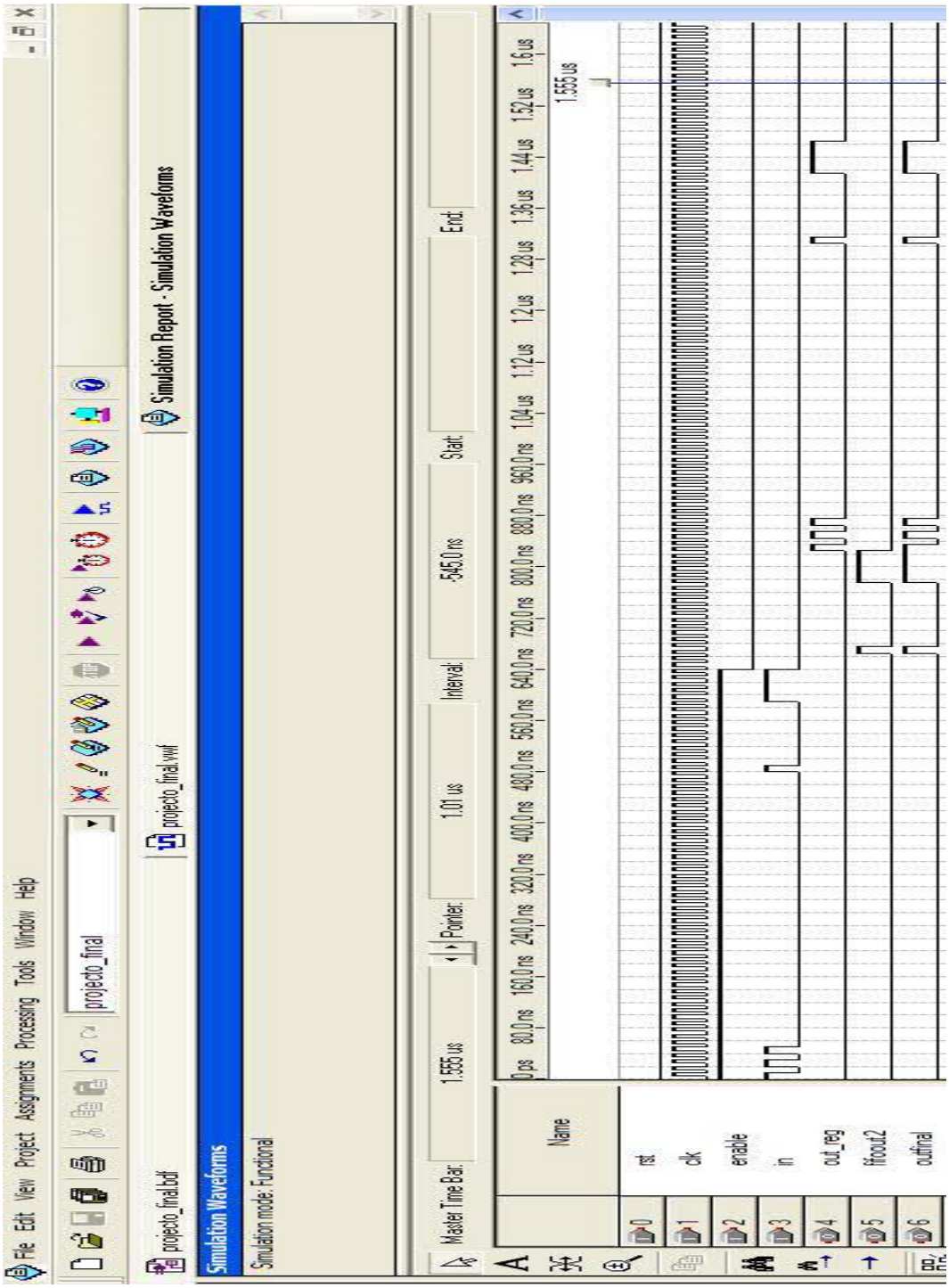

Figura 66: Resultados funcionais do bloco or no intervalo [0;1,6] µs

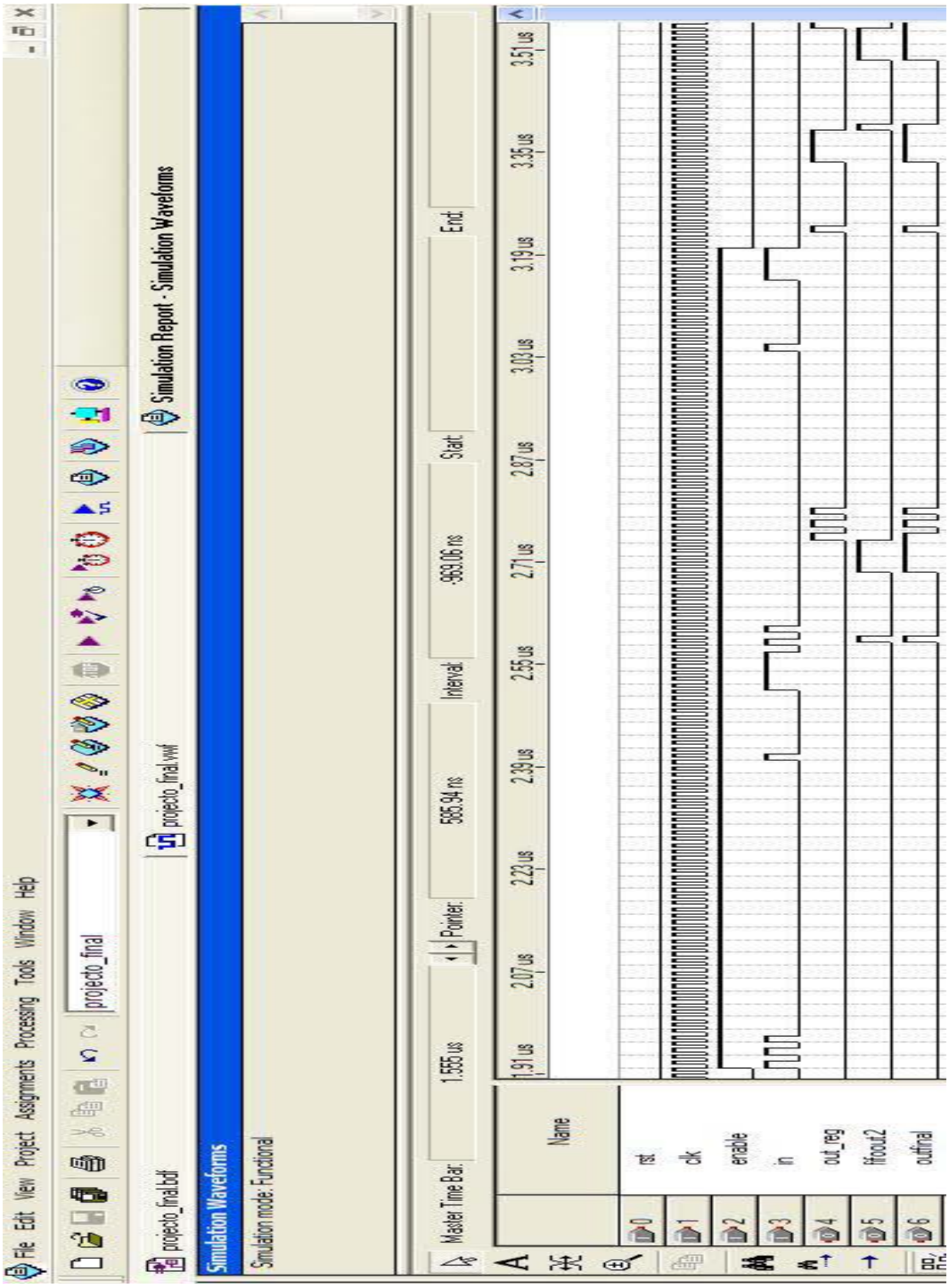

Figura 67: Resultados funcionais do bloco or no intervalo [1,91;3,51] µs

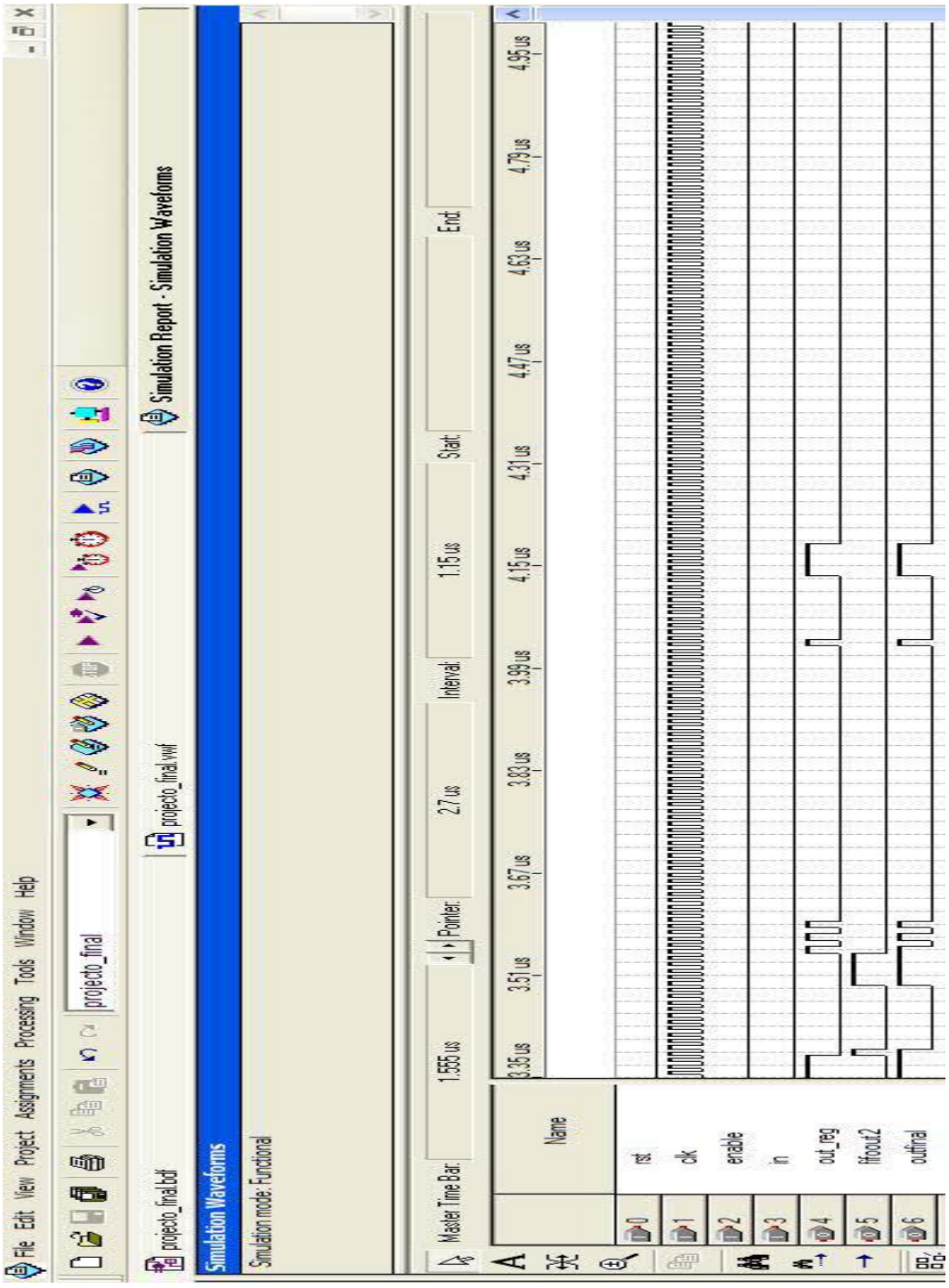

Figura 68: Resultados funcionais do bloco or no intervalo [3,35;4,95] µs

Capítulo V – Conclusões e Trabalho Futuro

5.1 − Conclusões

Tendo em conta as necessidades da evolução de novas técnicas de prevenção rodoviária através de novas tecnologias, para o benefício de toda a população quer a nível pessoal quer a nível socioeconómico, o presente trabalho teve como principal objectivo o desenvolvimento e implementação de um sistema em linguagem VHDL que permitisse a adição de uma extensão cíclica entre os blocos OFDM.

A presente dissertação descreve o desenvolvimento do sistema referido bem como as suas partes constituintes. Verificou-se através de simulações funcionais que foi possível implementar o sistema, bem como, foi possível constatar que existem várias maneiras diferentes de implementá-lo. Esta característica é inerente à linguagem VHDL, pois este tipo de linguagem, traduz-se numa grande liberdade de maneiras de atingir o mesmo objectivo por caminhos diferentes. Sendo assim este sistema final foi sendo aperfeiçoado ao longo do tempo, no qual, fui melhorando os meus níveis de conhecimento da linguagem VHDL e do software Quartus II.

Depois da conceptualização do sistema, o sistema foi implementado de forma gradual desde o bloco selector até ao or, pelo que inicialmente o sistema possuía mais blocos e usava mais recursos lógicos mas ao longo do tempo a implementação foi sendo optimizada, também devido à aprendizagem adquirida.

5.2 − Trabalho futuro

Como trabalho futuro fica a nota de que o sistema pode ser sempre ainda mais optimizado principalmente a nível dos fifos e recursos usados. Especificamente pode ser melhorada a parte em que o prefixo é escrito no fifo2, ou seja, conforme o prefixo é escrito no fifo2 pode também ser indo lido e neste momento está implementado como sendo todo escrito (16 bits) e no final é que é lido. Também a maneira como está a ser gerado o sinal de read2 no fifo_control2 pode ser feito de outra forma em vez de usar demais contadores e flag. Uma possível maneira seria colocar o bloco lpm_shiftreg entre os blocos fifo_control e fifo sendo aí gerado o atraso de 80 bits. Isto significaria que o sinal read2 no bloco fifo_control2 seria apenas um sinal negado do read1 do bloco fifo_control, tirando a ressalva de que o sinal read2 teria que ser adaptado para a leitura do primeiro bloco sendo depois apenas a negação do sinal read1. Estas alterações produziriam uma diminuição no tempo de processamento bem como na complexidade geral do sistema, tornando-o um pouco mais compreensível.

Bibliografia

[1] http://www.automedia.com/Accident_Prevention_Technology/dsm20030801ap/1 05/06/2009

[2] Stibor Lothar;Zang Yunpeng;"Neighborhood evaluation of vehicular ad-hoc network using IEEE 802.11p"; Aachen University.

[3] Fuertes Luis Zarzo; “OFDM PHY layer implementation based on the 802.11a standard and system performance analysis”; Linkoping University, 2005

[4] IEEE std 802.11a-1999(R2003);”Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications High-speed Physical Layer in the 5 GHz Band”, 2003

[5] Intini Aníbal Luis; “Orthogonal Frequency Division Multiplexing for Wireless Networks”; University of California Santa Barbara, 2000

[6] Dunbar Barry; “Software Simulator Development for Orthogonal Frequency Division Multiplexing (OFDM) Modulation”; University of Southern Queensland Faculty of Engineering & Surveying, 2006

[7] Abbasi Shahbaz; Baig Shazer; “Hardware Implementation of OFDM Transmitter and Receiver using FPGA”; National University of Computer and Emerging Sciences, 2008

[8] Lawrey Eric Phillip;”Adaptive Techniques for Multiuser OFDM”; Electrical and Computer Engineering School of Engineering James Cook University, 2001

[9] Adzha Kadiran Kamaru; “Design and Implementation of OFDM transmitter and receiver on FPGA Hardware”; Universiti Teknologi Malaysia, 2005

[10] http://www.leearmstrong.com/Dsrc/DSRCHomeset.htm 05/06/2009

[11] http://pt.wikipedia.org/wiki/VHDL 05/06/2009

[12] http://pt.wikipedia.org/wiki/Fpga 05/06/2009

[13] http://www.altera.com/corporate/crp-index.html 05/06/2009

[14] http://en.wikipedia.org/wiki/Altera 05/06/2009

[15] Marcelo Lima Duarte José; “Demodulador PM Digital para o sistema Brasileiro de Coleta de Dados ”; Universidade Federal do Rio Grande do Norte – Centro de Tecnologia, 2007

ANEXOS

Anexo A – bloco selector

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use std.textio.all;

entity selector is

  port
    (
      clk  : in std_logic;
      rst  : in std_logic;
      enable : in std_logic;
      data_in : in std_logic;

      sel1  : out std_logic;
      sel2  : out std_logic;
      data_out  : out std_logic

      );

end selector;

architecture selector_arch of selector is

signal contador : integer := 0;

begin
process (rst,clk)
begin

          if (rst = '1') then

      contador  <=   0;
      data_out <= '0';
      sel1 <= '0';
      sel2 <= '0';

    elsif rising_edge(clk) then

                  if (enable = '1') then

                                sel1 <= '1';
```

```
                    data_out  <= data_in;
                    contador  <= contador + 1;

                if (contador >= 48) then

                    if (contador = 64) then

                        sel2 <= '0';

                    else sel2 <= '1';

                    end if;

                end if;

                if (contador = 64) then

                    contador <= 1;

                end if;

            else

                if (contador = 64) then

                    contador <= 0;

                end if;

                sel1 <= '0';
                sel2 <= '0';
                data_out  <= '0';

            end if;

        end if;

end process;

end selector_arch;
```

Anexo B – bloco fifo_control

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use std.textio.all;

entity fifo_control is

  port
    (
      clk  : in std_logic;
      rst  : in std_logic;
      data_in : in std_logic;
      data_in_valid : in std_logic;

      write_enable : out std_logic;
      read_enable  : out std_logic;
      data_out  : out std_logic

      );

end fifo_control;

architecture fifo_control_arch of fifo_control is

signal s_contador, s_contador_in, s_contador_out : integer := 0;

begin
process (rst,clk)
begin

      if (rst = '1') then

                  s_contador  <= 0;
                  read_enable <= '0';
                  write_enable <= '0';
                  s_contador_in <= 0;
                  s_contador_out <= 0;

      elsif rising_edge(clk) then
```

```
if (data_in_valid = '1') then

        write_enable <= '1';
        data_out  <= data_in;
        s_contador  <= s_contador + 1;
        s_contador_in <= s_contador_in + 1;

    if (s_contador >= 64) then

        if (s_contador >= 80) then

            read_enable <= '1';
            s_contador_out <= s_contador_out + 1;
            s_contador <= 1;

        else

            read_enable <= '0';

        end if;

    else

    read_enable <= '1';
    s_contador_out <= s_contador_out + 1;

  end if;

else

    write_enable <= '0';
    data_out <= '0';
    s_contador  <= s_contador + 1;

    if (s_contador >= 64) then

        read_enable <= '0';

        if (s_contador >= 80) then

            read_enable <= '1';
```

```
                                s_contador_out <= s_contador_out + 1;
                                s_contador <= 1;

                        else

                                read_enable <= '0';

                        end if;

                else

                read_enable <= '1';
                s_contador_out <= s_contador_out + 1;

            end if;

                if (s_contador_out = s_contador_in) then

                        read_enable <= '0';
                        s_contador_in <= 0;
                        s_contador_out <= 0;
                        s_contador <= 0;

                end if;

            end if;

        end if;

end process;

end fifo_control_arch;
```

Anexo C – bloco fifo_control2

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use std.textio.all;

entity fifo_control2 is

  port
    (
      clk  : in std_logic;
      rst  : in std_logic;
      data_in : in std_logic;
      data_in_valid : in std_logic;
      data_in_valid2 : in std_logic;

      write_enable : out std_logic;
      read_enable  : out std_logic;
      data_out  : out std_logic

      );

end fifo_control2;

architecture fifo_control2_arch of fifo_control2 is

signal s_cont, s_contador_in, s_contador_out : integer := 0;
signal flag, s_read_enable : std_logic := '0';
signal s_contador_read_enable : integer := 0;
signal s_cont_data_in_valid2: integer := 0;

begin
process(s_read_enable, rst)
begin

        if (rst = '1') then

                s_contador_read_enable <= 0;

        else
```

```
                if rising_edge(s_read_enable) then
                        s_contador_read_enable <= s_contador_read_enable + 1;
                end if;
                if (flag = '0') then
                        s_contador_read_enable <= 0;
                end if;
        end if;
end process;
process (rst,clk)
begin
    if (rst = '1') then

                read_enable <= '0';
                s_read_enable <= '0';
                write_enable <= '0';
                s_contador_in <= 0;
                s_contador_out <= 0;
                s_cont <= 0;

          elsif rising_edge(clk) then
                if (data_in_valid = '1') then
                        write_enable <= '1';
                        data_out  <= data_in;
                        s_contador_in <= s_contador_in + 1;
                   if (flag = '1') then
                                s_cont <= s_cont + 1;
```

```
                if (s_cont >= 64) and (s_cont <= 80) then

                        read_enable <= '1';
                        s_read_enable <= '0';
                        s_contador_out <= s_contador_out + 1;

                        if (s_cont = 80) then

                                s_cont <= 1;
                                read_enable <= '0';
                                s_read_enable <= '1';

                        end if;


                end if;

        else

                s_cont <= 0;

        end if;

else

        data_out <= '0';
        write_enable <= '0';

        if (flag = '1') then

                s_cont <= s_cont + 1;

                if (s_cont >= 64) and (s_cont <= 80) then

                        read_enable <= '1';
                        s_read_enable <= '0';
                        s_contador_out <= s_contador_out + 1;

                        if (s_cont = 80) then

                                s_cont <= 1;
```

```
                                        read_enable <= '0';
                                        s_read_enable <= '1';

                                    end if;

                                end if;

                            else

                                s_cont <= 0;

                            end if;

                            if (s_contador_out = s_contador_in) then

                                s_contador_in <= 0;
                                s_contador_out <= 0;

                            end if;

                        end if;

                end if;

end process;

process(rst,data_in_valid2)
begin

        if (rst = '1') then

                flag <= '0';

        else

                if rising_edge(data_in_valid2) then

                        flag <= '1';

                end if;
```

```
                if (s_cont_data_in_valid2 = (s_contador_read_enable * 64)) and
(s_cont_data_in_valid2 > 0) then

                        flag <= '0';

                end if;

        end if;

end process;

process(rst,clk)
begin

        if (rst = '1') then

                s_cont_data_in_valid2 <= 0;

        else

                if rising_edge(clk) then

                        if (data_in_valid2 = '1') then

                                s_cont_data_in_valid2 <= s_cont_data_in_valid2 + 1;

                        end if;

                        if (flag = '0') then

                                s_cont_data_in_valid2 <= 0;

                        end if;

                end if;

        end if;

end process;

end fifo_control2_arch;
```

Anexo D – bloco FIFO

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

entity FIFO is

  port
     (

       DATAIN      : in std_logic;
       WRITE       : in std_logic;
       READ        : in std_logic;
       RESET       : in std_logic;
       CLK         : in std_logic;

       FULL        : out std_logic;
       EMPTY       : out std_logic;
       DATAOUT     : out std_logic

     );
end FIFO;



architecture BEHAVIOR of FIFO is

begin

  START: process

  type FIFO_TYPE is array (0 to 16384) of std_logic;
  variable FIFO: FIFO_TYPE;
  variable BOTTOM,TOP: INTEGER range 0 to 16384;
  variable C_SIZE:INTEGER range 0 to 16384;
  variable FLAG_EMPTY, FLAG_FULL: std_logic;

begin

  wait until CLK'EVENT and CLK = '1';
```

```
        if (RESET = '1') then

                FLAG_EMPTY := '1';
                FLAG_FULL  := '0';
                FULL  <= '0';
                EMPTY <= '1';
                TOP := 0;
                BOTTOM := 0;
                C_SIZE := 0;
                DATAOUT <= '0';

        end if;

if ( (WRITE = '1')  AND (C_SIZE <= 16384) AND (FLAG_FULL /= '1') ) then

   FIFO(BOTTOM) := DATAIN;
   BOTTOM := BOTTOM+1;

   if (BOTTOM > 16384) then

      BOTTOM := 0;

   end if;

   if ( C_SIZE < 16384 ) then

      C_SIZE := C_SIZE +1;

   else

      FULL <= '1';
      FLAG_FULL := '1';

   end if;

    EMPTY <= '0';
    FLAG_EMPTY := '0';

end if;
```

```
  if (READ='1') AND (C_SIZE > 0) AND (FLAG_EMPTY /= '1') then

     DATAOUT <= FIFO(TOP);
     TOP := TOP +1;

      if (TOP = 16384) then

        TOP := 0;

      end if;

      C_SIZE := C_SIZE -1;

      if ( C_SIZE = 0 ) then

         EMPTY <= '1';
         FLAG_EMPTY := '1';
         BOTTOM :=0;
         TOP :=0;

     end if;

     FULL <= '0';
     FLAG_FULL := '0';

  else

          DATAOUT <= '0';

  end if;

end process START;

end BEHAVIOR;
```

Anexo E – bloco FIFO2

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

entity FIFO2 is

  port
     (

       DATAIN      : in std_logic;
       WRITE       : in std_logic;
       READ        : in std_logic;
       RESET       : in std_logic;
       CLK         : in std_logic;

       FULL        : out std_logic;
       EMPTY       : out std_logic;
       DATAOUT     : out std_logic

     );

end FIFO2;

architecture BEHAVIOR of FIFO2 is

begin
  START: process

  type FIFO_TYPE is array (0 to 3300) of std_logic;
  variable FIFO: FIFO_TYPE;
  variable BOTTOM,TOP: INTEGER range 0 to 3300;
  variable C_SIZE:INTEGER range 0 to 3300;
  variable FLAG_EMPTY, FLAG_FULL: std_logic;

begin

  wait until CLK'EVENT and CLK = '1';

  if (RESET = '1') then
```

```
   FLAG_EMPTY := '1';
   FLAG_FULL  := '0';
   FULL  <= '0';
   EMPTY <= '1';
   TOP := 0;
   BOTTOM := 0;
   C_SIZE := 0;
   DATAOUT <= '0';

 end if;

if ( (WRITE = '1')  AND (C_SIZE <= 3300) AND (FLAG_FULL /= '1') ) then

   FIFO(BOTTOM) := DATAIN;
   BOTTOM := BOTTOM+1;

   if (BOTTOM > 3300) then

      BOTTOM := 0;

   end if;

   if ( C_SIZE < 3300) then

      C_SIZE := C_SIZE +1;

   else

      FULL <= '1';
      FLAG_FULL := '1';

   end if;

    EMPTY <= '0';
    FLAG_EMPTY := '0';

end if;

if (READ='1') AND (C_SIZE > 0) AND (FLAG_EMPTY /= '1') then

         DATAOUT <= FIFO(TOP);
```

```
        TOP := TOP +1;


    if (TOP = 3300) then

      TOP := 0;

    end if;

    C_SIZE := C_SIZE -1;

    if ( C_SIZE = 0 ) then

       EMPTY <= '1';
       FLAG_EMPTY := '1';
       BOTTOM :=0;
       TOP :=0;

   end if;

   FULL <= '0';
   FLAG_FULL := '0';

 else

        DATAOUT <= '0';

 end if;

end process START;

end BEHAVIOR;
```

Anexo F – bloco LPM_SHIFTREG

```
--      LPM_SHIFTREG Parameterized Megafunction (read only)
--
--  Copyright 1991-163849 Corporation
--  Your use of Altera Corporation's design tools, logic functions
--  and other software and tools, and its AMPP partner logic
--  functions, and any output files from any of the foregoing
--  (including device programming or simulation files), and any
--  associated documentation or information are expressly subject
--  to the terms and conditions of the Altera Program License
--  Subscription Agreement, Altera MegaCore Function License
--  Agreement, or other applicable license agreement, including,
--  without limitation, that your use is for the sole purpose of
--  programming logic devices manufactured by Altera and sold by
--  Altera or its authorized distributors.  Please refer to the
--  applicable agreement for further details.
--
--  9.0 Build 132  02/25/163849
--
--      Version 1.0
--
---------------------------------------------------------------------

INCLUDE "lpm_constant.inc";
INCLUDE "dffeea.inc";

PARAMETERS
(
        LPM_WIDTH,
        LPM_DIRECTION = "LEFT",
        LPM_AVALUE = "UNUSED",
        LPM_SVALUE = "UNUSED",
        DEVICE_FAMILY
);

INCLUDE "aglobal90.inc";

SUBDESIGN lpm_shiftreg
(
```

```
        data[LPM_WIDTH-1..0]                : INPUT = GND;
        clock                               : INPUT;
        enable                              : INPUT = VCC;
        shiftin                             : INPUT = VCC;
        load                                : INPUT = GND;
        aclr, aset                          : INPUT = GND;
        sclr, sset                          : INPUT = GND;
        q[LPM_WIDTH-1..0]                   : OUTPUT;
        shiftout                                    : OUTPUT;
)

VARIABLE
        IF (FAMILY_HAS_PRESET() == 0 & USED(aclr) & USED(aset))
GENERATE
                dffs[LPM_WIDTH-1..0]        : dffeea;
        ELSE GENERATE
                dffs[LPM_WIDTH-1..0]        : DFFE;
        END GENERATE;

        shift_node[LPM_WIDTH-1..0]      : NODE;
        IF (USED(LPM_AVALUE)) GENERATE
                ac      : lpm_constant with (LPM_WIDTH=LPM_WIDTH,
LPM_CVALUE=LPM_AVALUE);
        END GENERATE;
        IF (USED(LPM_SVALUE)) GENERATE
                sc      : lpm_constant with (LPM_WIDTH=LPM_WIDTH,
LPM_CVALUE=LPM_SVALUE);
        END GENERATE;

BEGIN

        ASSERT (LPM_WIDTH > 0)
                        REPORT "Value of LPM_WIDTH parameter must be greater
than 0"
                        SEVERITY ERROR
                        HELP_ID LPM_SHIFTREG_WIDTH;

        ASSERT (USED(aset) # USED(LPM_AVALUE) == 0)
                        REPORT "Ignored LPM_AVALUE parameter because the aset
port is not used"
                        SEVERITY WARNING
                        HELP_ID LPM_SHIFTREG_AVALUE;
```

```
	ASSERT (USED(sset) # USED(LPM_SVALUE) == 0)
			REPORT "Ignored LPM_SVALUE parameter because the sset
port is not used"
			SEVERITY WARNING
			HELP_ID LPM_SHIFTREG_SVALUE;

	ASSERT (LPM_DIRECTION == "LEFT" # LPM_DIRECTION == "RIGHT")
			REPORT "Illegal value for LPM_DIRECTION parameter (%) --
value must be LEFT or RIGHT"
				LPM_DIRECTION
			SEVERITY ERROR
			HELP_ID LPM_SHIFTREG_DIRECTION;

	ASSERT (FAMILY_IS_KNOWN() == 1)
			REPORT "Megafunction lpm_shiftreg does not recognize the
current device family (%) -- ensure that you are using the newest version of the
megafunction"
				DEVICE_FAMILY
			SEVERITY WARNING
			HELP_ID LPM_SHIFTREG_FAMILY_UNKNOWN;

	% common ports %
	dffs[].ena = enable;
	dffs[].clk = clock;

	% Asynchronous control logic %
	IF (USED(LPM_AVALUE)) GENERATE
		dffs[].clrn = !aclr & (!aset # ac.result[]);
		dffs[].prn  = aclr # !aset # !ac.result[];
	ELSE GENERATE
		IF (USED(aclr)) GENERATE
			dffs[].clrn = !aclr;
		END GENERATE;
		IF (USED(aset)) GENERATE
			dffs[].prn = aclr # !aset;
		END GENERATE;
	END GENERATE;

	% Shift direction nodes %
	if (LPM_WIDTH > 1) GENERATE
		IF (LPM_DIRECTION == "LEFT") GENERATE
```

```
                shift_node[] = (dffs[LPM_WIDTH-2..0].q, shiftin);
                shiftout = dffs[LPM_WIDTH-1].q;
            ELSE GENERATE
                shift_node[] = (shiftin, dffs[LPM_WIDTH-1..1].q);
                shiftout = dffs[0].q;
            END GENERATE;
        ELSE GENERATE
            shift_node[] = shiftin;
            shiftout = dffs[0].q;
        END GENERATE;

        % Synchronous input logic %
        IF (USED(LPM_SVALUE)) GENERATE
            dffs[].d = !sclr & (  sset & sc.result[]
                                                # !sset & ( !load & shift_node[]
                                                                # load & data[]));
        ELSE GENERATE
            dffs[].d = !sclr & (sset # ( !load & shift_node[]
                                                            # load & data[]));
        END GENERATE;

        % Connect outputs %
        q[] = dffs[].q;
        IF !USED(q) GENERATE
            q[] = GND;
        END GENERATE;
        IF !USED(shiftout) GENERATE
            shiftout = GND;
        END GENERATE;
END;
```